NASA Technical Memorandum 100574

THE ENVIRONMENT FOR APPLICATION SOFTWARE INTEGRATION AND EXECUTION (EASIE) VERSION 1.0

VOLUME II

PROGRAM INTEGRATION GUIDE

KENNIE H. JONES
DONALD P. RANDALL
SCOTT S. STALLCUP
LAWRENCE F. ROWELL

DECEMBER 1988

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23665-5225

PREFACE

The Environment for Application Software Integration and Execution (EASIE) provides both a methodology and a set of software utility programs to ease the task of coordinating engineering design and analysis codes. The need for such techniques and tools has stemmed from the computer-aided design and engineering activities within Langley Research Center's Space Systems Division (SSD). In SSD, the Vehicle Analysis Branch (VAB), with emphasis on advanced transportation systems, and the Spacecraft Analysis Branch (SAB), with emphasis on advanced spacecraft, share a common need to integrate many stand-alone engineering analysis programs into coordinated, quick-turnaround, user-friendly design systems. In particular, the most needed capabilities include easy selection of application programs, quick review and modification of program input/output data, and logging of the actual steps that were executed during the study. Although the application programs used by VAB and SAB differ, the design methods used by their engineers are quite similar, and great efficiency can be gained by providing a computer environment that provides the capabilities mentioned above.

EASIE is a user interface and set of utility programs which supports rapid integration and execution of programs about a central relational database. In general, the EASIE system addresses the needs of four different classes of people who will

be involved in the development of an engineering design system. Certain individuals may serve in more than one of these roles, but the following terms will help to clarify several distinct activities associated with the EASIE system.

The first classification represents the engineer/designer/analyst. This group conducts the design study by executing modeling and analysis programs and generating data required to evaluate the design against its objectives. EASIE documentation will refer to this group as EASIE system users or, more often, as users. In general, these users are only interested in executing programs already installed into an EASIE design system.

A second group aided by EASIE is identified as application programmers. These programmers are responsible for the development and improvement of modeling and analysis programs used in the engineering design process. They are the experts with respect to particular application programs and can define its input and output variables. This must be done before inclusion of that program with others into the integrated system.

The third group is identified as program implementers since their function is to provide an environment where all the software tools work together with a minimum of effort. These people will use information provided by the application programmers and will install or modify the programs in an EASIE system by creating appropriate data constructs in the database and locating files where needed by the EASIE executive.

The fourth classification is design team leader or design manager. This is the individual or group responsible for identifying parameters important to the design study and for configuration management of the data as it is produced by the design team. This design manager must have an overview of the total data requirements for the analysis process and must be concerned foremost with the integrity of the data.

With these terms defined, the four volumes of EASIE documentation can be associated with the groups most likely to use them. Each of the volumes addresses different aspects of the support tools, and each is intended to be independent of the others.

Volume I, EXECUTIVE OVERVIEW, provides information about the functions, concepts, and historical development of EASIE and should be read by anyone trying to determine if EASIE would be beneficial to their work.

Volume II, PROGRAM INTEGRATION GUIDE, describes the portion of the EASIE tools supporting both the integration of application programs into a central database and the definition of the data dictionary used during data review and modification. This volume will be used primarily by the program implementer and the design manager in their responsibilities for the actual installation of appropriate programs into a fully-integrated design system. However, the application programmer may also use tools described in this volume to assist in the documentation of input/output variables for the application program.

Volume III, PROGRAM EXECUTION GUIDE, describes the portion of the EASIE tools supporting the selection and execution of application programs, building of menus, and editing of program data. This volume will be of foremost importance to the users who will perform design studies. In addition, the program implementers will find the sections concerning the construction of application-dependent procedures helpful. Finally this document will also be used by the design manager for reviewing data and design activities.

Volume IV, SYSTEM INSTALLATION AND MAINTENANCE GUIDE, describes the procedure of loading the EASIE system onto a computer. It also gives some insight into the hardware and software dependencies of the EASIE code. This, most likely, will be needed by the program implementer to familiarize himself with the directory structure and location of the various EASIE components. Although the design of EASIE is intended to reduce the system dependencies, this version nevertheless reflects in several ways the current implementation using the Relational Information Management (RIM*) database management system and the VAX/VMS+ operating system.

* Trademark of Boeing Computer Services

\+ Trademark of the Digital Equipment Corporation

TABLE OF CONTENTS

TABLE OF CONTENTS

LIST OF FIGURES

1.0 INTRODUCTION

The purpose of this document is to present a set of software tools designed to aid in the complex task of integrating a collection of application programs into a single system. The Environment for Application Software Integration and Execution (EASIE) tools offer a program integration approach critical in defining and forming the data paths for inter-program communication.

This document is intended for programmers faced with the problems of program integration or program interfacing. For readers unfamiliar with the rigors of program integration, the following BACKGROUND section provides the necessary motivation. Although the acronym EASIE implies simplicity, the problems encountered when integrating two or more programs are often formidable. Consequently, the potential EASIE user should anticipate a relatively high learning curve upon initial exposure to the database management system (DBMS) terminology and software tools. One of the principal advantages of using the EASIE tools is that once the initial learning hurdle is conquered, the learned program integration techniques are applicable to other programs or sets of programs.

In addition to a knowledge of the program integration problem, the reader should have some experience with DBMS terminology and techniques. In particular, the EASIE tools rely heavily on the techniques of the relational approach to DBMS such as those described by C. J. DATE [1]. More specifically, the EASIE tools are built upon the Relational Information Management

(RIM) DBMS [2]. However, as explained later in this document, the EASIE tools are DBMS independent.

The computing environment in which the EASIE tools were developed is a DEC VAX 11/785 running VMS 4.5. In some sections of this document, necessary references are made to VMS utilities, symbols, and other DEC system features. The interested reader should consult DEC documentation for further details. The source language is DEC's version of FORTRAN 77. Because certain EASIE utilities involve code generation in the form of FORTRAN subroutines, the prospective user should be familiar with FORTRAN language constructs.

The first few sections of this document provide background information on the program integration problem. The majority of this document is organized as a tutorial whereby new terms and concepts are introduced as they are encountered in the normal course of the program integration process. A SIMPLE INTEGRATION EXAMPLE section introduces the basic features of the EASIE tools, defines the relevant terminology, illustrates the menu and command driven user interface, and concludes with program execution. EASIE provides an environment for both program integration and execution. As the name signifies, A MORE COMPLEX EXAMPLE section details additional features of the EASIE DBMS tools using a more sophisticated program integration problem. The last two sections are reference guides to the SYSTEM LIBRARY PROCESSOR Menu Options and to the REVIEWER commands including ranges, limitiations, capabilities, and defaults.

The architecture of the EASIE user interfaces has been designed, as much as possible, to be independent of the database management system (DBMS) software and computer hardware. EASIE Volume IV, SYSTEM INSTALLATION AND MAINTENANCE GUIDE [3], describes the considerations important to porting the EASIE system to other computer systems using other DBMS software. EASIE Volume III, PROGRAM EXECUTION GUIDE [4], can be used independent of the other EASIE Volumes, and provides the guidelines for the selection and execution of application programs, building menus, reviewing data, and editing program input data. EASIE Volume I, EXECUTIVE OVERVIEW [5], contains the information that is useful in determining if EASIE would be beneficial to a specific project or study.

2.0 BACKGROUND

Over the past 25 years, the use of computer aids in engineering has increased exponentially. Today, much work that a few years ago might have taken weeks can be done in seconds. Application programs are available for virtually all engineering disciplines.

This growth has not been void of problems. While information may be obtained at a faster rate and in greater abundance, the absence of standardization and lack of coordination among software developers has resulted in the proliferation of computer programs unable to communicate data to other programs. Although two programs may use the same data, physical restrictions imposed by the computer environment may inhibit the communication of the data between the programs. The price being paid for this state of affairs is often the manual transfer of data from one program to another with the associated manpower loss, time delay, and potential for error introduction. Several approaches can be taken to address this problem. This document describes a set of software tools designed to implement one approach used at LaRC for its efficiency and flexibility. A brief example will serve to compare these approaches.

Assume a modeling program (PROGA) has been purchased from Company A and an analysis program (PROGB) from Company B. Among the output from PROGA is the TOTAL MODEL MASS written to an unformatted file. PROGB requires the TOTAL MODEL MASS for input, but the program is designed to read it from a formatted file.

Logically, these programs could communicate data (TOTAL MODEL MASS), though physically this communication is impossible due to differing data environments (formatted versus unformatted files). Consequently, some change must be made to enable the transfer of data. This is typical of the problems faced when several programs are to share data within a design activity.

At least three possible methods exist to facilitate communication between programs under such circumstances:

(1) Modify one or both programs (program integration).

(2) Develop a translator to convert the output of PROGA into an acceptable input format for PROGB (program interfacing).

(3) Use a database management system (DBMS) to store the output from PROGA into the database and retrieve the input for PROGB from the database.

For this example, method (1) appears to be the simplest, most efficient means to communicate data. PROGA can be modified to output TOTAL MODEL MASS in a format acceptable to PROGB. However, the real world is seldom so simple. In a realistic example, PROGA would likely produce abundant output of which only a portion would be required by PROGB. Format conversion would not be the only problem to solve. All possible output statements would have to be located and understood before the format conversion could be completed. In a large, complex program, this would not be a trivial task.

The scenario can be worse. Suppose PROGA is to supply inputs to several programs in the integrated system, each requiring data in different formats. PROGA would require modifications to output, possibly the same data, to several output files (additional overhead).

Although program complexity may render this approach impractical, purchasing agreements may make it impossible. Many software products are acquired without access to source code and therefore may not be modified.

For this example, method (2) solves some of these problems. Although the internal structure of programs is often not readily accessible (complex code or no code), the inputs to and outputs from programs are usually well documented. A translator program could be developed to read as input the output of PROGA and write an output file acceptable as input to PROGB. This method has the advantage of requiring no modification to either program, but introduces additional overhead as it reads, converts, and writes. Also, a separate translator would be required for each application.

Method (3) may be implemented as a variation of either method (1) or method (2). Using method (3), data are retrieved from and stored in a relational database rather than converted directly to another format. Using the simplified example, PROGA could be modified to store TOTAL MODEL MASS directly into a relational database; or if no source code is available, a translator could be developed to read the output file from PROGA and store TOTAL MODEL MASS in the database. PROGB could then retrieve TOTAL MODEL MASS from the database by using either method (1) or (2). In some cases, method (3) imposes additional overhead because the data must be manipulated by the DBMS being used and programmers with experience in using the selected DBMS are required. However, there are significant advantages to be

gained when program integration is accomplished through method (3). A DBMS is a collection of tools designed for fast random access storage and retrieval of data allowing organization and referencing of data by logical relationships without concern for the physical file organization. If such tools are used exclusively in program integration, they establish standards for the physical storage of data and reduce communication problems associated with the data environment (differing formats).

Because of their random access capabilities, the DBMS tools may increase efficiency in cases where small amounts of data are required from a large output. Suppose PROGA outputs TOTAL MODEL MASS at the end of a large output file. By method (2), if PROGB and PROGC required TOTAL MODEL MASS, translators for both would have to process the entire output file to locate the datum. Using a DBMS, after the output from PROGA is stored in the database, TOTAL MODEL MASS may be retrieved quickly without unproductive processing. But most important, any new programs added to the design activity that require output from PROGA can access that datum without requiring further modification to PROGA or its translator.

Although the advantages provided through method (3) are attractive, especially when numerous programs are to be coordinated, many programmers are hesitant to use a DBMS because of the additional knowledge required. Not only are programmers experienced with the interface to the DBMS required, but the portability of the resultant software is reduced. If the software is to be moved to another computer, the new machine must

have the selected DBMS. Conversions to other DBMS's are usually costly and time consuming. Even conversions from one computer to another, using the same DBMS, may require substantial software changes.

An alternative is required that retains the advantages of using a DBMS while minimizing the impact on the program integrator. The EASIE data management tools are designed to offer such an alternative. EASIE does not eliminate the need for knowledge of a relational approach to database management, but does reduce the need for specific knowledge of the interface to the selected DBMS. These tools, which are addressed in the remainder of this document, are based upon a processor which makes the DBMS transparent to the user, while providing the functions needed to quickly integrate (interface) programs around a central database. The speed and versatility of the DBMS approach make it the most productive method for the development of a system of integrated programs.

3.0 THE EASIE TOOLS FOR DATABASE MANAGEMENT AND PROGRAM INTEGRATION

The EASIE DBMS tools allow programmers to work at a higher level of abstraction than that provided through conventional DBMS's. Using a user-friendly, interactive processor, the schema of the database is described (relation names and descriptions; attribute names and descriptions; data types; dimensions; and units). Next, the data (program input or output) required for each particular application are identified in an input or output template. This template information, referred to as the SYSTEM LIBRARY, is stored in the database and aids in the automation of several tasks in the integration effort. Having described the required relations, the initial database schema is automatically produced. Using a generic editor, referred to as the REVIEWER, and the template specification for the application program, a consistent systematic method is provided to review/modify the input or review the output for any program. Also, using the template specification, FORTRAN subroutines are generated automatically to retrieve data (for an input template) from the database into a program's local variables. Similarly, subroutines are generated to store data (for an output template) from local variables into the database.

Using the EASIE tools, conventional tasks for database integration are substantially reduced, program maintenance is simplified, and the program integration task is more straightforward. In the following sections of this document, a simple example using EASIE is presented providing the minimum discussion needed to complete the example. A later section presents a more

complex example using additional EASIE techniques. The final sections provide a complete reference guide to the EASIE tools.

3.1 Parameter Versus Attribute

Conventionally, a relational DBMS allows for organization of data into tables, called relations, consisting of rows (tuples) and columns (attributes) which provide storage for the matrix of data. Most DBMS's do not provide a convenient means for storing parametric data (data not representable by a matrix). For example, in a modeling system, mass and volume of parts may be represented by a table:

Part Name	Mass	Volume
TOP	10	100
MIDDLE	20	200
BOTTOM	30	300

A database relation representing this table may be defined with attributes: part name, mass, and volume. Each part would have a tuple entry giving the actual values for each part. But how would the total model mass and total model volume be stored? Here a relation (matrix) representation is not appropriate. EASIE provides for a special relation type defined as PARAMETER type. A parameter relation is a collection of parameters of varying types and dimensions that logically belong together. Thus, a parameter relation could be defined as follows:

Parameter name	Value
TOTMASS	60
TOTVOL	600

to contain the values for total model mass (TOTMASS) and total model volume (TOTVOL).

In EASIE, the conventional matrix relation is called ATTRIBUTE type.

3.2 Integration Using The Conventional Approach

To integrate a program with a DBMS, the following steps must be taken:

(1) Examine the program and identify all data to be read from the database (input) or written to the database (output).

(2) Group the data into relations according to logical relationships among the data accounting for various types, dimensions, etc. Organize the retrieval/output of data from/to the database required by each program.

(3) Using the DBMS, create a database containing the schema for the required relations.

(4) Using the FORTRAN interface library of the DBMS, write FORTRAN code to retrieve input from the database and store output into the database.

(5) Because the interactive data modification capabilities available with the DBMS are often not suitable for efficient data modification, a FORTRAN processor may be required to facilitate input modification and output review. This processor may be called from within the program (integrated) or executed as pre- (input review) or post- (output review) processor to the program (interfaced).

Often steps (3), (4), and (5) are the most difficult, time-consuming, and error-prone activities. They also represent the portion of this process that requires specific knowledge of the selected DBMS. Using the EASIE tools, these steps are automated to the extent that schemata, FORTRAN database I/O routines, and the review capabilities are produced without the need for such knowledge.

3.3 Integration Using EASIE

Within EASIE, information about the database and program interaction with the database is recorded using an interactive program, the SYSTEM LIBRARY PROCESSOR. A data dictionary is constructed containing the name and description of each relation; and the name, type, dimension, description, and units of each parameter/attribute of the relation. A template library is constructed containing the input and output templates required by programs integrated with the database. A template is a description of all relation subsets required as input/output to/from a program. A program library is constructed storing information about each program integrated into the system; i.e., location, execution procedure, input and output templates, etc. All of this information is stored in a RIM database, The SYSTEM LIBRARY.

Once this information is recorded in the SYSTEM LIBRARY database, the SYSTEM LIBRARY PROCESSOR can automatically produce:

(1) an input file or FORTRAN program to create the schema for the database with which the programs will be integrated.

(2) FORTRAN subroutines to retrieve input and store output from/to the database for each program (using the input and output templates). Data are retrieved/stored from/in the database into/from variables in the subroutines that have the same names as the parameters/attributes in the target relation. All communication code using the FORTRAN interface to the DBMS is constructed by the processor.

(3) an input file to a generic REVIEWER program. The file consists of an input or output template and instructs the REVIEWER on which information to extract from the database for review or modification. Therefore, the one REVIEWER program may be used to retrieve or modify data for any application.

After creating the schema for the database using output from the processor, default input data may be established using the REVIEWER providing one or more Master databases [6] exist and which may be copied and modified. The generated FORTRAN routines to retrieve/store data must be modified by the integrator to communicate data to/from the program by:

(1) adding parameter and/or common blocks to the subroutine to pass the values from/to the variables declared by the processor. Also, the routine calling statement must be placed at the appropriate location in the applications program (program integration).

(2) creating a preprocessor (for input templates) or post-processor (for output templates). The routines are modified to create a stand-alone program that, after reading the database, will write an input file for the program or, after reading the output file, store the data in the database (program interfacing).

In either case, database interaction has been automated.

4.0 A SIMPLE INTEGRATION EXAMPLE

The following is a step-by-step example to illustrate the integration of a program with a database. Suppose a program, BOX, is to be integrated with a database. BOX accepts as input the length, width, and height of a box and computes its volume.

Step 1: Length, width, and height (real numbers of dimension 1) are required as input in units of meters(M). Volume (real number of dimension 1) is output in cubic meters (M**3). To aid in the process of defining relations, a standard form is suggested (Appendix F).

Step 2: All values are parameter type (represent no row/column relationship) and will be placed in a single relation DIMEN. Before proceeding to steps (3), (4), and (5), the SYSTEM LIBRARY must be constructed.

4.1 Constructing The SYSTEM LIBRARY

The SYSTEM LIBRARY is physically a RIM database and the schema must be established before entering data. The SYSTEM LIBRARY database is initially created by executing the VMS command:

```
@TOAIDE:[BUILD_DICT]BUILDDICT.COM<CR>
```

(See Appendix G for definition of symbols and logicals in VMS). This places in the current directory a RIM database named DICT containing the schema for the SYSTEM LIBRARY. The RIM database consists of three files: DICT1.DAT, DICT2.DAT, and DICT3.DAT.

The SYSTEM LIBRARY PROCESSOR is executed in the same directory containing the SYSTEM LIBRARY database by the command:

```
RUNDICT<CR>
```

resulting in the display of the following prompt.

```
ENTER NUMBER OF CHARACTER/WORD FOR TARGET SYSTEM:
      4 FOR RIM ON PRIME/VAX
     10 FOR RIM ON NOS
      2 FOR SDRC/PRL
```

In this example, FORTRAN code is generated to interact with a RIM database on a VAX computer. Therefore, the selected response is 4.

The next display reveals the processor's main menu.

```
SELECT OPTION:
BR  - TO BUILD RELATIONS
AR  - TO ADD TO AN EXISTING RELATION
BT  - TO BUILD A TEMPLATE
BF  - TO BUILD FORMATTER ROUTINES
BS  - TO BUILD A SCHEMA DUMP FOR DATA BASE
BRV - TO BUILD A REVIEWER INPUT FILE
BP  - TO BUILD PROGRAM DESCRIPTION
PT  - TO PRINT A TEMPLATE
LR  - TO LIST RELATIONS
PR  - TO PRINT A RELATION
X   - TO EXIT
```

4.2 Defining The Data Dictionary

The first task is to define the required relations (only DIMEN in this example). Thus BR is selected, resulting in a series of queries to define a relation.

```
ENTER RELATION NAME (OR <CR> WHEN DONE):
```

Enter the relation name "DIMEN" <CR>.

```
ENTER 1 FOR PARAMETER TYPE
   OR 2 FOR ATTRIBUTE TYPE
```

DIMEN is to be parameter type (enter 1 <CR>).

```
ENTER RELATION DESCRIPTION (80 CHARACTERS):
```

The relation description is used to later identify contents of a relation. Enter BOX DIMENSIONS.

```
ENTER YOUR NAME (20 CHARACTERS):
```

The name of the relation creator may be useful at a later time as new programs are integrated. Enter the creator's name (the current date is also recorded). Each parameter in the relation DIMEN must now be described following the prompt.

DEFINE THE RELATION DIMEN:

The information necessary to describe a parameter is requested. Recall that the first parameter is named LENGTH, of type REAL, of dimension 1, and is measured in meters M. The parameter description is defined as BOX LENGTH. The following sequence of prompts and user responses (underlined) define the parameter length.

```
ENTER PARAMETER NAME (OR <CR> WHEN DONE)>
LENGTH<CR>
ENTER PARAMETER TYPE>
R<CR>
ENTER PARAMETER DIMENSION (UP TO 3 SEPARATED BY COMMAS)>
1<CR>
DIMENSION = 1
ENTER PARAMETER DESCRIPTION>
BOX LENGTH<CR>
ENTER UNIT OF MEASUREMENT (<CR> IF N/A)>
M<CR>
ENTER OUTPUT FORMAT (<CR> IF N/A)>
<CR>
OK TO ENTER THE PARAMETER LENGTH    INTO THE DATABASE
(Y/N) (<CR>=N):
Y<CR>
```

Output format is entered as a <CR> to indicate that the default format for this type (real) is to be used by the REVIEWER. Following the last entry, the sequence is repeated for width (WIDTH, R, 1, BOX WIDTH, M, <CR>, Y), height (HEIGHT, R, 1, BOX HEIGHT, M, <CR>, Y), and volume (VOLUME, R, 1, BOX VOLUME, M**3, <CR>, Y). If a mistake is made during the sequence, an N on the final response will eliminate the entire parameter definition from the DATA DICTIONARY, and the sequence for that parameter may be reentered. When the first prompt of the sequence appears following the definition of the final parameter/attribute, by entering a <CR> will end the relation DIMEN definition and the

following prompt will be repeated:

ENTER RELATION NAME (OR <CR> WHEN DONE):

The entry of another name would allow the definition of another relation. Because the example has only the relation DIMEN, a <CR> will end the DATA DICTIONARY definition and return to the main menu.

When a good working knowledge of EASIE is achieved, an implementer of a program requiring a large amount of input data, should use the BUILD_EASIE utility described in Appendix E.

4.3 Defining The Template Library

The second step is to define the TEMPLATE LIBRARY. Two templates are required for the application program BOX. The input template (named BOXIN) in effect makes the statement, as input to program BOX, LENGTH, WIDTH, and HEIGHT are to be retrieved from the relation DIMEN. Similarly, the output template (named BOXOUT) in effect makes the statement, as output from program BOX, VOLUME is to be stored in the relation DIMEN.

The main menu command, BT, is selected resulting in a series of queries to define a template.

ENTER TEMPLATE NAME (OR <CR> WHEN DONE):

Enter the input template name BOXIN.

ENTER 1 IF INPUT TEMPLATE
OR 2 IF OUTPUT TEMPLATE

BOXIN, an input template, defines data to be retrieved from the database for input to the program BOX. The selected choice is 1. Following the prompt:

IDENTIFY ALL INPUT RELATIONS FOR THE TEMPLATE

the information describing data to be retrieved is requested. Recall that input to the program BOX consisted of LENGTH, WIDTH, and HEIGHT from the relation DIMEN. Following the prompt:

```
ENTER RELATION NAME (OR <CR> WHEN DONE):
```

the name DIMEN is entered. Because DIMEN was previously defined as a parameter type relation, the response to the prompt

```
ENTER 1 FOR PARAMETER TYPE
   OR 2 FOR ATTRIBUTE TYPE
```

is 1 and the opportunity to review its parameters is provided by the following prompt:

```
THE RELATION DIMEN  IS ALREADY DEFINED AND HAS    4 PARAMETERS.
      DO YOU WANT TO SEE THEM (Y/N) (<CR>=N):
```

An affirmative response, Y, results in the following display depicting necessary parameter information:

```
PARAMETER NAME    DIM1   DIM2   DIM3   TYPE   NUMBER OF CHARACTERS
LENGTH                                 REAL
WIDTH                                  REAL
HEIGHT                                 REAL
VOLUME                                 REAL
```

Following this display or a negative response, N, the next prompt provides the names of parameters to be retrieved.

```
ENTER 0 IF ALL PARAMETERS ARE TO BE RETRIEVED
   OR 1 TO READ PARAMETERS TO BE RETRIEVED FROM A FILE
   OR 2 TO ENTER PARAMETERS TO BE RETRIEVED FROM THE TERMINAL
```

A response of 2 allows input of the desired parameters by prompting,

```
ENTER PARAMETER NAMES EACH FOLLOWED BY <CR>:
(PARAMETER NAME=<CR> WHEN DONE)
```

Recall that only LENGTH, WIDTH, and HEIGHT are required from DIMEN as input. A response of,

```
LENGTH<CR>
WIDTH<CR>
```

HEIGHT<CR>
<CR>

ends the selection of parameters from the relation DIMEN and the inquiry,

```
YOU HAVE IDENTIFIED       3 PARAMETERS TO BE RETRIEVED
OK TO ENTER THE RELATION DIMEN      INTO THE TEMPLATE BOXIN
(Y/N) (<CR>=N):
```

allows those selections to be entered into the template BOXIN (affirmative response).

At this point, parameter/attributes from other relations could be added to the template BOXIN following the repeated prompt.

```
ENTER RELATION NAME (OR <CR> WHEN DONE):
```

As no other relations are required, a <CR> completes the series of queries for defining a template and iterates the series for subsequent template definitions. The following sequence illustrates the definition of the output template BOXOUT (user responses are underlined):

```
ENTER TEMPLATE NAME (OR <CR> WHEN DONE):
BOXOUT<CR>
ENTER 1 IF INPUT TEMPLATE
   OR 2 IF OUTPUT TEMPLATE
2<CR>
IDENTIFY ALL OUTPUT RELATIONS FOR THE TEMPLATE

ENTER RELATION NAME (OR <CR> WHEN DONE):
DIMEN<CR>
ENTER 1 FOR PARAMETER TYPE
   OR 2 FOR ATTRIBUTE TYPE
1<CR>
THE RELATION DIMEN     IS ALREADY DEFINED AND HAS          4
PARAMETERS.
DO YOU WANT TO SEE THEM (Y/N) (<CR>=N):
N<CR>
ENTER 0 IF ALL PARAMETERS ARE TO BE REPLACED
   OR 1 TO READ PARAMETERS TO BE REPLACED  FROM A FILE
   OR 2 TO ENTER PARAMETERS TO BE REPLACED  FROM THE
TERMINAL
2<CR>
```

```
ENTER PARAMETER NAMES EACH FOLLOWED BY <CR>:
(PARAMETER NAME=<CR> WHEN DONE)
VOLUME<CR>
<CR>
YOU HAVE IDENTIFIED        1 PARAMETERS TO BE REPLACED
OK TO ENTER THE RELATION DIMEN      INTO THE TEMPLATE
BOXOUT    (Y/N) (<CR>=N):
Y<CR>
ENTER RELATION NAME (OR <CR> WHEN DONE):
<CR>
ENTER TEMPLATE NAME (OR <CR> WHEN DONE):
<CR>
```

The final <CR> returns to the processor's main menu marking the completion of the TEMPLATE LIBRARY definition.

STEP 3:

4.4 Constructing The Database Schema

Having completed the DATA DICTIONARY and TEMPLATE LIBRARY descriptions, an input file to RIM must be created for database schema definition. The command BS allows the option for either of two DBMS's.

```
ENTER 1 FOR RIM SCHEMA
   OR 2 FOR SDRC/PRL SCHEMA
```

Because RIM is the DBMS for this example, 1 is entered. The name of a file to contain the schema definition is supplied following the prompt.

```
ENTER SCHEMA DUMP FILE NAME
OR "Q" TO RETURN WITHOUT WRITING A FILE
```

Entering SCHEMA.DAT (name selection is arbitrary) results in a file of that name in the current directory containing the schema definition. The schema for a single relation may be written to the file by naming the relation following the prompt:

```
ENTER RELATION NAME FOR SCHEMA DUMP
   OR <CR> FOR COMPLETE DATABASE SCHEMA DUMP
   OR "Q" TO QUIT
```

The usual response (as in this example) is to create the schema for the complete database by entering a <CR>.

Outside the EASIE processor, interactive RIM must be executed by issuing the following single commands (APPENDIX G),

```
RUNRIM
INPUT SCHEMA<CR>
```

which creates the schema for the database under the default name DATADB.

STEP 4:

4.5 Producing FORTRAN Routines For Input And Output To/From The Database

Most DBMS's provide a FORTRAN interface library of subroutines to access the database. Data are transferred from/to a relation one row (tuple) at a time using a one-dimensional array. If the attribute types of the relation differ (integer, real, character, etc.), an integer and real array must be equivalenced for storage and packing/unpacking of the storage array with the data in the order of the attributes in the relation. (Character data may be packed into either the integer or real array.) In addition to packing/unpacking the storage array, other functions must be performed by these FORTRAN routines. The database must be opened. The relations to be accessed must be identified, and the search or sort conditions for locating the data of interest must be established. The DBMS error codes and row counts must be controlled. Without automatic procedures to generate such code, a substantial amount of code would need to be written by the program integration team and then

tested and proven correct, often a time-consuming activity. Frequently, this new code is scattered throughout the existing code, making future maintenance efforts more difficult.

Using the BF command, an input or output template is used to automatically produce FORTRAN code for database interaction. Data are either retrieved from the database and stored into program variables (input template) or extracted from program variables and stored in the database (output template). FORTRAN declarative statements (name, type, and dimension) for these program variables are automatically generated by the processor. Variable names produced by the processor match the names of the parameters or attributes as they exist in the database. The only responsibility of the program integration team is to communicate these variables to the program.

After issuing the BF command, the selected template name is supplied following the prompt:

```
ENTER TEMPLATE NAME
OR <CR> FOR ALL RELATIONS
OR "Q" TO QUIT
```

To produce input routines for this example, BOXIN is entered. Next, the file name to contain the input subroutines is provided following the prompt:

```
ENTER GETDATA FILE NAME
OR "Q" TO RETURN WITHOUT WRITING A FILE.
```

Arbitrarily, BOXIN.FOR is entered. The next prompt:

```
DO YOU REQUIRE ANSI STANDARD CODE? (Y/N) (<CR>=N)
```

provides the opportunity to reduce eight character parameter/attribute names in the data dictionary to six characters (adhering to ANSI standards). This is done by

truncation of any characters beyond six and may result in variable name conflict. Because a VAX will accept eight character variable names, the answer is N. The final prompt:

```
DO YOU WANT TO RENAME ANY OTHER PARAMETERS/ATTRIBUTES?
(Y/N) (<CR>=N)
```

allows variable names to be different from parameter/attribute names in the data dictionary. Suppose the predefined parameter named LENGTH was to be assigned to an existing variable named L in the program. An affirmative answer to this inquiry would provide the opportunity to assign the value of LENGTH in the database into the variable L instead of the variable LENGTH. In this example, the parameter names are satisfactory for variable names. The answer is N, and the processor returns to the main menu.

Once produced, the file BOXIN.FOR contains source code for a driver subroutine (GETDATA_) and for one subroutine for each relation to be accessed (1 in this case). The driver subroutine GETDATA_ is the only routine that need be modified by the program implementer and may be seen in figure 1. Notice that the variables LENGTH, WIDTH, and HEIGHT, are all typed REAL with no dimension, and the database is given the default name of DATADB, reference the statement DBASE_=(8HDATADB). After the call to the subroutine R1001_, the variables LENGTH, WIDTH, and HEIGHT will contain the values in the database for the parameters LENGTH, WIDTH, and HEIGHT. The only changes required by the program integrator are those necessary to communicate the variables back to the program via either arguments lists or common blocks. The parameters could be added to the CALL and

SUBROUTINE statements for the GETDATA_ subroutine. An alternative, and the method used in this example, is to add a common block containing the variables LENGTH, WIDTH, HEIGHT, and VOLUME to the GETDATA_ subroutine under the assumption that this common block also exists in the calling program.

```
      SUBROUTINE GETDATA_
C
      COMMON /DBNAME_/ DBASE_,DBOPN_
      REAL*8 DBASE_
      LOGICAL DBOPN_
C
C*** THE PARAMETERS FOR THE RELATION DIMEN    ARE:
      REAL     LENGTH
      REAL     WIDTH
      REAL     HEIGHT
C
C   LOAD REQUIRED COMMON BLOCKS HERE:
C
      INCLUDE 'BOXIN.COMMON/LIST'
      DBASE_ = (8HDATADB  )
C
      CALL R1001_(LENGTH,WIDTH,HEIGHT)
C
C   MAKE ANY NECESSARY RE-ASSIGNMENTS HERE:
C
      INCLUDE 'BOXIN.ASSIGN/LIST'
C
      RETURN
C
      END
```

Figure 1. - Subroutine GETDATA_

Notice the comment,

```
C   LOAD REQUIRED COMMON BLOCKS HERE:.
```

Directly below is the VAX FORTRAN INCLUDE statement added by the implementer to the automatically-generated code. To add common blocks, the program integrator need only create a file using the name BOXIN.COMMON containing the required common blocks. No

changes are required in the file BOXIN.FOR. Thus a file named BOXIN.COMMON is then created containing the common block:

```
COMMON /INDIMEN/ LENGTH,WIDTH,HEIGHT.
```

Also notice the comment,

```
C   MAKE ANY NECESSARY RE-ASSIGNMENTS HERE:
```

following the call to the SUBROUTINE R1001_. If, for example, the database contained LENGTH in meters and the program expected input of LENGTH in feet, an assignment statement could be added at this point to make the proper unit conversion. As this is not the case in this example, no code additions are required here, and the statement,

```
INCLUDE 'BOXIN.ASSIGN/LIST'
```

should be deleted. The file BOXIN.FOR is now ready for compilation and linkage with the example program.

The BF command must be re-executed for the output template BOXOUT. The following sequence of prompts and user responses (underlined) build the file BOXOUT.FOR containing the FORTRAN routines required to store VOLUME into the database,

```
BF<CR>
ENTER TEMPLATE NAME
OR <CR> FOR ALL RELATIONS
OR "Q" TO QUIT
BOXOUT<CR>
ENTER PUTDATA FILE NAME
OR "Q" TO RETURN WITHOUT WRITING A FILE
BOXOUT.FOR<CR>
DO YOU REQUIRE ANSI STANDARD CODE? (Y/N) (<CR>=N)
N<CR>
DO YOU WANT TO RENAME ANY OTHER PARAMETERS/ATTRIBUTES?
(Y/N) (<CR>=N)
N<CR>
```

Similar to the file BOXIN.FOR, BOXOUT.FOR contains a driver SUBROUTINE PUTDATA_ and one subroutine for each relation to be

accessed (1 in this case). The SUBROUTINE PUTDATA_ may be seen in figure 2.

```
      SUBROUTINE PUTDATA_
C
      COMMON /DBNAME_/ DBASE_,DBOPN_
      REAL*8 DBASE
      LOGICAL DBOPN_
C
C*** THE PARAMETERS FOR THE RELATION DIMEN    ARE:
      REAL     VOLUME
C
C    LOAD REQUIRED COMMON BLOCKS HERE:
C
      INCLUDE 'BOXOUT.COMMON/LIST'
      DBASE_ = (8HDATADB  )
C
C    MAKE ANY NECESSARY RE-ASSIGNMENTS HERE:
C
      INCLUDE 'BOXOUT.ASSIGN/LIST'
C
      CALL W1001_(VOLUME)
C
      RETURN
C
      END
```

Figure 2. - Subroutine PUTDATA_

Notice that the variable VOLUME is typed REAL with no dimension, and again the default database name DATADB is assigned to the variable DBASE_. Noting the FORTRAN INCLUDE statement, the program implementer need only create a file named BOXOUT.COMMON containing the statement:

COMMON /OUTDIMEN/ VOLUME.

The common variable VOLUME is assumed to contain the proper value upon entry to the SUBROUTINE PUTDATA_. Any necessary format changes or other reassignments need to be made prior to the call to the SUBROUTINE W1001_. As none are required here, the statement,

```
INCLUDE 'BOXOUT.ASSIGN/LIST'
```

should be deleted. Immediately following the call to W1001_, the value of VOLUME is stored in the database. With the creation of the file BOXOUT.COMMON, the file BOXOUT.FOR is ready for compilation and linkage with the example program.

4.6 Linking The Example Program

The original program BOX is seen in figure 3A. Notice that length, width, and height are read from the file BOXIN.DAT and Volume is written to the file BOXOUT.DAT.

```
PROGRAM BOX
COMMON /INDIMEN/ LENGTH,WIDTH,HEIGHT
REAL LENGTH
COMMON /OUTDIMEN/ VOLUME
OPEN(8,FILE='BOXIN.DAT')
READ(8,*)LENGTH,WIDTH,HEIGHT
CLOSE(8)
VOLUME = LENGTH * WIDTH * HEIGHT
OPEN(9,FILE='BOXOUT.DAT')
WRITE(9,*)VOLUME
CLOSE(9)
END
```

Figure 3A.- Original Program Box

The modified example program (contained in the file BOX.FOR) is seen in figure 3B.

```
      PROGRAM BOX
      COMMON /INDIMEN/ LENGTH,WIDTH,HEIGHT
      REAL LENGTH
      COMMON /OUTDIMEN/ VOLUME
C
      CALL GETDATA_
C
      VOLUME = LENGTH * WIDTH * HEIGHT
C
      CALL PUTDATA_
C
      END
```

Figure 3B.- Modified Program Box

Notice that former input data transfer statements (FORTRAN OPEN, READ, and CLOSE) are replaced by a single call to the SUBROUTINE GETDATA__. Former output data transfer statements (OPEN, WRITE, and CLOSE) are replaced by a single call to the SUBROUTINE PUTDATA__. Both common blocks (INDIMEN and OUTDIMEN) have been included. Following the call to GETDATA (input of LENGTH, WIDTH, and HEIGHT), VOLUME is computed and, through the call to PUTDATA__, stored in the database. The file BOX.FOR is compiled and linked via the following command sequence (created in the BOX.COM file), generating the executable file BOX.EXE.

```
$FOR BOX.FOR
$FOR BOXIN.FOR
$FOR BOXOUT.FOR
$LINK BOX.OBJ -
    + BOXIN.OBJ -
    + BOXOUT.OBJ -
    + LOADRIM/LIBRARY
$EXIT
```

STEP 5:

4.7 Creating REVIEWER Input Files

To this point, the DATA DICTIONARY (DICT) and TEMPLATE LIBRARY (BOXIN, BOXOUT) have been created. Using them, a database schema, DATADB, and FORTRAN input/output routines (BOXIN.FOR, BOXOUT.FOR) have been created. These routines have been linked with the example program BOX which is now ready for execution by obtaining its input from the database DATADB. However, no values for LENGTH, WIDTH, and HEIGHT have been established in the database. Execution at this point would retrieve the value of zero (by default) for all three values resulting in a computation of zero for volume.

Step (5) of the integration using the conventional approach involves the use of the EASIE processor called the REVIEWER.

Using the BRV command, a template is used to create an input file for a standard REVIEWER program. Using the REVIEWER, input values are changed and output values are examined.

After entering the command BRV, the prompt:

```
ENTER DIRECTORY LOCATION FOR REVIEWER INPUT FILES
OR <CR> TO CREATE A LOCAL REVIEWER FILE
```

allows the user to specify a directory location for the REVIEWER input file. This is important when using the directory structure recommended for use [6,7] with the EASIE EXECUTIVE. For this example, the REVIEWER input file remains in the current directory by entering a carriage return. In response to the prompt:

```
ENTER TEMPLATE NAME
   OR <CR> TO CREATE REVIEWER FILES FOR ALL TEMPLATES
   OR "Q" TO QUIT
```

the template name BOXIN is supplied. A file is thus created in the current directory under the name BOXIN.REV containing input directing the REVIEWER to retrieve LENGTH, WIDTH, and HEIGHT from the relation DIMEN. The REVIEWER allows user modification and replacement of the data.

Similarly, the following sequence of prompts and user responses (underlined) create a file in the current directory under the name BOXOUT.REV containing input directing the REVIEWER to retrieve VOLUME for review:

```
BRV<CR>
ENTER DIRECTORY LOCATION FOR REVIEWER INPUT FILES
OR <CR> TO CREATE A LOCAL REVIEWER FILE
<CR>
ENTER TEMPLATE NAME
   OR <CR> TO CREATE REVIEWER FILES FOR ALL TEMPLATES
   OR "Q" TO QUIT
BOXOUT<CR>
```

4.8 Executing The Example Program

Having created both the input file for the REVIEWER (BOXIN.REV) and the executable file (BOX.EXE), input to the program BOX may be modified in the database by executing the command:

REVIEW BOXIN<CR>.

This will execute the REVIEWER using as input the file BOXIN.REV. Immediately the display in the top half of figure 4 will appear. This demonstrates how the use of the SYSTEM LIBRARY PROCESSOR tools can automate most of the integration tasks even including an editing capability for data needed by the integrated programs.

Issuing the commands (descriptions in brackets are not a part of the command),

```
M 1 5<CR>       [MODIFY LENGTH TO 5 METERS]
M 2 10<CR>      [MODIFY WIDTH TO 10 METERS]
M 3 15<CR>      [MODIFY HEIGHT TO 15 METERS]
R<CR>           [REDRAW THE SCREEN]
```

will modify the values for LENGTH, WIDTH, and HEIGHT and redraw the screen with the new values. The final screen display is shown in the bottom half of figure 4. Finally, the command:

E<CR>

will replace the values in the database.

Execution of the example program will now retrieve these input values, compute VOLUME, and replace VOLUME in the database. The VMS command,

RUN BOX.EXE<CR>

will execute BOX.

Issuing the VMS command:

REVIEW BOXOUT<CR>

will execute the REVIEWER using as input the file BOXOUT.REV. Immediately the display in figure 5 will appear. Note the correct value of 750.000M^3 has been computed for the parameter VOLUME.

4.9 Concluding The Simple Example

The following is a review of the process for integrating the program BOX. In the previous example, the SYSTEM LIBRARY PROCESSOR was used to:

(1) Define a parameter relation (DIMEN).

(2) Define input (BOXIN) and output (BOXOUT) templates accessing some portion of the relation DIMEN.

(3) Build an input file for RIM to create the schema (SCHEMA.DAT) for DIMEN.

(4) Build FORTRAN subroutines (BOXIN.FOR and BOXOUT.FOR) to access portions of the database as described in the input and output templates.

(5) Build input files for the REVIEWER to review/modify data described by the input and output templates (BOXIN.REV and BOXOUT.REV).

After modifications to the program BOX.FOR (to call the FORTRAN routines mentioned in step 4 above) and modifications to the input and output subroutines (that pass data to and from BOX.FOR), the three files, BOX.FOR, BOXIN.FOR, and BOXOUT.FOR were compiled and linked. Executing the REVIEWER using the input template BOXIN to establish LENGTH, WIDTH, and HEIGHT, was followed by execution of BOX (BOX.EXE) which stored the resultant VOLUME into the database. The computed value for VOLUME could be reviewed through execution of the REVIEWER using the output

template BOXOUT.

To aid in the variable definition task, the EASIE Form 1 as explained in Appendix F may be useful. Notice that after designing the relations and templates (work that must be done in some form no matter what integration approach is used), the relations are placed in the DATA DICTIONARY. Anyone familiar with a relational DBMS (such as RIM) will recognize that this procedure, with the exception of defining parameter/attribute descriptions and units, is not more difficult than the conventional approach to define the database schema. Defining the templates may be argued to be additional work, but the procedure is simple and once completed much of the remaining difficult work required by the conventional approach is automated. Production of the schema is accomplished by instructing the SYSTEM LIBRARY PROCESSOR to create an input file which is read by RIM. Routines to interact with the database using the FORTRAN interface library of RIM are automatically produced leaving the program implementer responsible for the task of defining the FORTRAN communications of data from one subroutine to another (no special RIM knowledge required). Input to the generic REVIEWER program is automatically produced for any given template, and data can be reviewed or modified using the same techniques for any template, thus for any program. Recall that all steps but the actual execution are performed once for each program by the integration team. The designer/engineer performs the final steps of data review, data modification, and program execution.

Obviously, many of the tedious, error-prone tasks required

using the conventional approach have been eliminated. However, other advantages may not yet be apparent. Because the FORTRAN I/O routines are produced automatically and data modification is accomplished using the REVIEWER, potentially any relational DBMS may be used. Originally, the EASIE tools were implemented using A Relational Information Management System (ARIS). Later, because of changing requirements at Langley Research Center, the tools were converted to use RIM. As needs changed, the capability to use PEARL (Structural Dynamics Research Corporation's I-DEAS system [8]) was added. During these transitions, no changes were required in the EASIE integrated programs. As the desired DBMS was changed, only new FORTRAN I/O routines and REVIEWER input files needed to be generated using the modified SYSTEM LIBRARY PROCESSOR. Since the REVIEWER uses the same commands and presents the data the same, there is no impact in this area regardless of the selected template or DBMS.

Also, future maintenance efforts have been reduced. Anyone having worked on a large system of programs integrated into a DBMS using the conventional approach understands how such a system can create maintenance problems. Database interaction code written by a large group of programmers may be scattered throughout many I/O processors and programs. Coding standards may be established, but are difficult to enforce. Using EASIE, coding standards are established and enforced by using the SYSTEM LIBRARY PROCESSOR. Hand-written I/O processors are replaced by the REVIEWER. I/O subroutines are written by the processor, thus eliminating the need for a maintenance programmer to read or

write database interaction code.

Conventional DBMS's usually provide a means to "dump" data to an ASCII file for data transfer from one database to another. If the data are logically the same but the schemata are different, conventional dump utilities cannot accommodate different schemata. Using EASIE, FORTRAN routines can be created to retrieve from one database and store in another. The programmer need only assign logically identical data from one variable to another.

```
                              CATEGORY        1:     DIMEN
BOX DIMENSIONS

-------------------------------------------------------------------------------
 L  !   PRESENT VALUE  !   NAME    !  SUBSCRIPT   ! DESCRIPTION       ! UNITS
-------------------------------------------------------------------------------
  1 !        0.0000000 ! LENGTH    !              ! BOX LENGTH        ! M
  2 !        0.0000000 ! WIDTH     !              ! BOX WIDTH         ! M
  3 !        0.0000000 ! HEIGHT    !              ! BOX HEIGHT        ! M
-------------------------------------------------------------------------------

M n : modify value (n = line#,name(subscript),or line range)
C n : change category (n = id or name)
N n : next page (n = + or - pages)
R : reprint page, L n : n line#'s per page, X n : expand line# n
E : end and save mods, Q : quit without saving mods, H : help
CAT : list categories, SUB : define review subset, T : toggle menu

EDIT:
>M 1 5<CR>
>M 2 10<CR>
>M 3 15<CR>
>R<CR>

                              CATEGORY        1:     DIMEN
BOX DIMENSIONS

-------------------------------------------------------------------------------
 L  !   PRESENT VALUE  !   NAME    !  SUBSCRIPT   ! DESCRIPTION       ! UNITS
-------------------------------------------------------------------------------
  1 !        5.0000000 ! LENGTH    !              ! BOX LENGTH        ! M
  2 !       10.0000000 ! WIDTH     !              ! BOX WIDTH         ! M
  3 !       15.0000000 ! HEIGHT    !              ! BOX HEIGHT        ! M
-------------------------------------------------------------------------------

M n : modify value (n = line#,name(subscript),or line range)
C n : change category (n = id or name)
N n : next page (n = + or - pages)
R : reprint page, L n : n line#'s per page, X n : expand line# n
E : end and save mods, Q : quit without saving mods, H : help
CAT : list categories, SUB : define review subset, T : toggle menu

EDIT:
>E<CR>
```

Figure 4. - REVIEWER Display of Length, Width, and Height

```
                                CATEGORY        1:     DIMEN
BOX DIMENSIONS

-------------------------------------------------------------------------------
 L  !   PRESENT VALUE  !   NAME    !  SUBSCRIPT   ! DESCRIPTION       ! UNITS
-------------------------------------------------------------------------------
  1 !       750.0000000 ! VOLUME    !              ! BOX VOLUME        ! M**3
-------------------------------------------------------------------------------

M n : modify value (n = line#,name(subscript),or line range)
C n : change category (n = id or name)
N n : next page (n = + or - pages)
R : reprint page, L n : n line#'s per page, X n : expand line# n
E : end and save mods, Q : quit without saving mods, H : help
CAT : list categories, SUB : define review subset, T : toggle menu

EDIT:
>Q<CR>
```

Figure 5. - REVIEWER Display of Volume

5.0 A MORE COMPLEX EXAMPLE

While the simple example does illustrate the major functions of the SYSTEM LIBRARY PROCESSOR and the standard flow of the program integration task, some additional capabilities available through the EASIE system may require further explanation. To accomplish this, a second example will be described. For the sake of brevity, the user prompts and proper responses are not embedded within the discussion, but rather are included in Appendix A. No step-by-step reference will be made to this appendix. It is highly desireable to follow along the printed screen text during the discussion.

In this example, the following new features will be demonstrated:

(1) The use of Attribute relations (in addition to Parameter relations).

(2) The use of character and integer data types and multi-dimensional data structures.

(3) Program interfacing in addition to program integration.

5.1 A Description Of The Programs

Suppose two commercially available programs require communication. The program MAKGEO is available with source code (Appendix B) and receives as its input the length, width, and height of a box. MAKGEO outputs the geometry for the box centered at the origin with length along the X-axis, width along the Z-axis, and height along the Y-axis. The geometry output consists of an array of point coordinates (X, Y, Z) and an array of faces. (For each face, start at $point_i$, draw to $point_j$, draw to $point_k$, draw to $point_l$ and draw to $point_i$, where i,j,k,l are

indices into the point array.) Because source code is available, user subroutines can be linked into the program to retrieve length, width, and height from the database and store the geometry into the database. This method, as in the simple example, represents program integration with the database.

The program DRAW is available only as an executable record, i.e. no FORTRAN source code. It is a generic display processor accepting as input a facet geometry description, i.e. points and faces as described above. Because no changes can be made to the input module of DRAW, an input file must be provided in the format (figure 6) acceptable to program DRAW. A preprocessor will be required to retrieve the geometry from the database and create a correctly formatted input file. The preprocessor must then be executed prior to execution of program DRAW. This method represents program interfacing with the database.

MODEL NAME	A80
X, Y, Z ROTATIONS IN RADIANS	3(E12.4)
NUMBER OF NODE POINTS	I5
X, Y, Z COORDINATES OF FIRST NODE	3(E12.4)
.	.
.	.
.	.
X, Y, Z COORDINATES OF LAST NODE	3(E12.4)
NUMBER OF FACES	I5
FACE(1),FACE(2),FACE(3),FACE(4)FOR FIRST FACE	4I5
.	.
.	.
.	.
FACE(1),FACE(2),FACE(3),FACE(4)FOR LAST FACE	4I5

Figure 6. - Format Of Input File For The Program DRAW

5.2 Defining The Relations

Since the variables length, width, and height (input to MAKGEO) have previously been defined in the relation DIMEN, they do not require redefinition. Two relations require definition to

contain the geometry (output from MAKGEO and input to DRAW.) These relations are defined using the BR command and include:

(1) NODES - An attribute relation with attributes (columns) X, Y, and Z (all real numbers, dimensioned 1 and measured in meters.) The three attributes contain the X, Y, and Z coordinates of the points. Each row then represents a three-dimensional point in space. The number of rows represents the number of points in the current model.

(2) FACES - an attribute relation with one attribute (column), FACE (an integer array dimensioned 4), containing the indices to the node relation rows comprising a facet in the model. Each row of the relation FACES represents a single face composed of moving to the FACE(1) row of nodes, drawing to FACE(2) row of nodes, drawing to FACE(3) row of nodes, drawing to FACE(4) row of nodes, and finally, closing the facet by drawing to FACE(1) row of nodes. Note that this relation could have been defined having 4 integer attributes each dimensioned 1 to contain the 4 indices to the NODES relation. The selected definition was chosen to illustrate EASIE's multidimensional capability for attribute relations.

Notice in figure 6, that the program DRAW requires as input a model name (1st record of the input file as a character string of length 80) and X, Y, and Z rotation angles for the model (2nd record of the input file as three real values). A parameter relation, MODEL, is defined to contain these two parameters.

5.3 Defining The Templates

Using the BT command, an input template, MAKGEOIN, is defined for the program MAKGEO. The parameters LENGTH, WIDTH, and HEIGHT are to be retrieved from the relation DIMEN.

During template creation, attribute relations present several options not applicable to parameter relations. In addition to selecting a subset of attributes, a subset of rows may be specified by entering one or more WHERE clauses, stating conditions required for row selection. Example: where x greater

than 500. Rows may be retrieved, sorted by selected attributes, by entering a SORT clause. In this example neither of the special capabilities is required. There are also several options for replacing rows. For more detail on these options, reference section 6.3.

An output template, MAKGEOOT, is defined for the program MAKGEO. The attributes X, Y, and Z, are to be stored in the relation NODES, using no special conditions and no sort conditions. The rows are to be replaced such that only rows in the current model remain in NODES. All others are deleted, using the option, replace all tuples exclusively. All attributes, FACE, are to be stored in the relation FACES in the same way.

An input template, DRAWIN, is defined for the program DRAW. All parameters are to be retrieved from the relation MODEL. All attributes are to be retrieved from the relations NODES and FACES using no special conditions and no sort conditions.

No output template is required for the program DRAW because its output is a graphical display of the contents of MODEL, NODES, and FACES.

5.4 Building The Database Schema

Using the RS command, an input file, called SCHEMA.DAT, to interactive RIM is created which contains the commands to build the schema for the relations DIMEN, NODES, FACES, and MODEL.

5.5 Building The FORTRAN I/O Subroutines

Through the BF command, the input template MAKGEOIN is used to generate the file MAKGEOIN.FOR which contains the subroutines to retrieve length, width, and height from the relation DIMEN. Similarly, the template MAKGEOOT is used to generate the file MAKEGEOOT.FOR which contains the subroutines to store the model node points into the relation NODES and the model faces into the relation FACES. Because the attributes for NODES and FACES will be represented as FORTRAN arrays, dimensions are supplied for the array declarations. These dimensions should match the dimensions within the program. Because MAKGEO constructs a box, 8 nodes and 6 faces are required. Finally, the template DRAWIN is used to generate the file DRAWIN.FOR which contains the subroutines to retrieve NAME and ROTATION from the relation MODEL, the model node points from the relation NODES, and the model faces from the relation FACES. Because the program DRAW is capable of displaying a model of up to 100 nodes and 100 faces, a dimension of 100 is selected for both NODES and FACES.

5.6 Building The REVIEWER Input Files

Prior to executing MAKGEO, values for LENGTH, WIDTH, and HEIGHT must be established in the database. To provide this capability, a REVIEWER input file is created (MAKEGEOIN.REV) using the template MAKGEOIN.

Prior to executing DRAW, values for NAME and ROTATION should be established in the database. This capability is provided by creating a REVIEWER input file using the template DRAWIN. Recall

that in addition to MODEL, the relations NODES and FACES are also accessed by the template DRAWIN. Assuming a decision is made that geometry modification should only be allowed through modifications to LENGTH, WIDTH, and HEIGHT and re-execution of MAKGEO, a row modification flag in the template input file is set to 2 (no row modification) for both NODES and FACES. Other flag values would allow modification to the geometry using the REVIEWER. An alternative would have been to create a template which only accessed the relation MODEL. However, using the selected method, a user may review the geometry as NAME and ROTATION are modified.

5.7 Preparing The Programs

Use of the SYSTEM LIBRARY PROCESSOR is complete. The remaining tasks to complete the integration are:

(1) Read the SCHEMA.DAT input command file into interactive RIM to create the database.

(2) Include common blocks in MAKGEOIN.FOR and MAKGEOOT.FOR to communicate data to/from the program MAKGEO.

(3) Modify MAKGEO to call the subroutines GETDATA_ and PUTDATA_ in the appropriate places.

(4) Link MAKGEO object file with objects of MAKGEOIN, MAKGEOOT and RIM library.

(5) Include FORTRAN code in the DRAWIN.FOR source file to create an input file that is used by program DRAW. This input file contains data retrieved from the database.

(6) Link DRAWIN object file and RIM library as a stand-alone program to be executed as a preprocessor to DRAW.

The schema for the database may be established by the command:

$RUNRIM

followed by the RIM command:

INPUT SCHEMA.

Within the FORTRAN source code for the program MAKGEO, (MAKGEO.FOR, Appendix B) are the common blocks:

COMMON/INDIMEN/LENGTH,WIDTH,HEIGHT

COMMON/OUTGEO/X(8),Y(8),Z(8),NG,FACE(4,6),NF.

A file named MAKGEOIN.COMMON must be created containing the common block INDIMEN. A file named MAKEGEOOT.COMMON must be created containing:

COMMON/OUTGEO/X,Y,Z,N002_,FACE,N003_.

Notice that dimensions for X, Y, Z, and FACE have been eliminated because they are contained in separate declarations within MAKGEOOT.FOR.

Also, notice in the call to SUBROUTINE W1002_ (passing the parameters to be stored in relation NODES) are the parameters X, Y, and Z which are arrays containing the node points and the integer parameter N002_ that must contain the number of node rows to be stored in the database. N002_ has then replaced NG in the common block OUTGEO. Similarly, in the call to SUBROUTINE W1003_, is the integer parameter N003_ which must contain the number of face rows to be stored in the database. N003_ has then replaced NF in the common block OUTGEO.

Since no variable reassignments are necessary, the code line:

INCLUDE 'MAKGEOIN.ASSIGN/LIST'

is eliminated (by commenting) from the file MAKGEOIN.FOR. Also the code line:

INCLUDE 'MAKGEOOT.ASSIGN/LIST'

is eliminated (by commenting) from the file MAKGEOOT.FOR. A recommended alternative approach is to create empty files using the names MAKGEOIN.ASSIGN and MAKGEOOT.ASSIGN thus preventing the need for the modifications to the files MAKGEOIN.FOR and MAKGEOOT.FOR.

A call to SUBROUTINE GETDATA_ (in file MAKGEOIN.FOR) is added to the beginning of the program MAKGEO, and a call to SUBROUTINE PUTDATA_ is added to the end of the program MAKGEO. MAKGEO.FOR, MAKGEOIN.FOR, and MAKGEOOT.FOR are compiled and linked using the file MAKGEO.COM.

Execution of MAKGEO will now retrieve LENGTH, WIDTH, and HEIGHT from the database (call to GETDATA_) passing these values via COMMON/INDIMEN/ to the main program which will fill the arrays X, Y, Z, and FACE, and the parameters NG and NF with the geometry for the specified box. The main program will pass the values via COMMON/OUTGEO/ to SUBROUTINE PUTDATA_ which will store the geometry into the relations NODES and FACES. Recall that this method represents program integration with the database.

Because the program DRAW is available as an executable file only, it cannot be modified. The file DRAWIN.FOR must be modified to become a stand-alone program that will retrieve the model name, rotations, and geometry from the database and format the data in a file as input to DRAW.

The SUBROUTINE, RETURN, and COMMON INCLUDE statements are commented.

The file DRAWIN.ASSIGN is created, amd contains the statements to write the model name, rotations, and geometry (in the required format) for the file DRAW.DAT, as input to the program DRAW.

Notice that the program DRAW requires rotations to be input in radians, but in this example, rotations are stored in the database as degrees and converted to radians prior to writing the DRAW.DAT file. DRAWIN.FOR is compilied and linked using the file DRAWIN.COM.

Execution of DRAWIN will now retrieve model name, rotations, and geometry from the database and create an input file DRAW.DAT to be used as input to the program DRAW. Recall that this method represents program interfacing with the database.

5.8 Execution Of The Complex Example

The VMS command: $REVIEW MAKGEOIN

produces the display shown in Appendix C.

The REVIEWER commands:

```
M LENGTH 1
M WIDTH  2
M HEIGHT 3
E
```

to establish LENGTH, WIDTH, and HEIGHT of the box as 1, 2, and 3 meters, respectively.

The VMS command: $RUN MAKGEO

executes MAKGEO and creates the box geometry.

The VMS command: $REVIEW DRAWIN

produces the display shown in Appendix D.

The REVIEWER commands:

```
M  1   'TEST BOX'
M  2   20
M  3   30
M  4   40
E
```

establishes the model name 'TEST BOX' and X, Y, and Z rotations of 20, 30, and 40 degrees, respectively.

The VMS command: $RUN DRAWIN

executes DRAWIN and creates the input file for the program DRAW.

The VMS command: $RUN DRAW

produces the display, shown in figure 7, only on a TEKTRONIX 4014 or compatible terminal. Source for the program DRAW is available on the file

TOAIDE:[EXAMPLE.SOURCE]DRAW.FOR

and may be modified to use another graphics system.

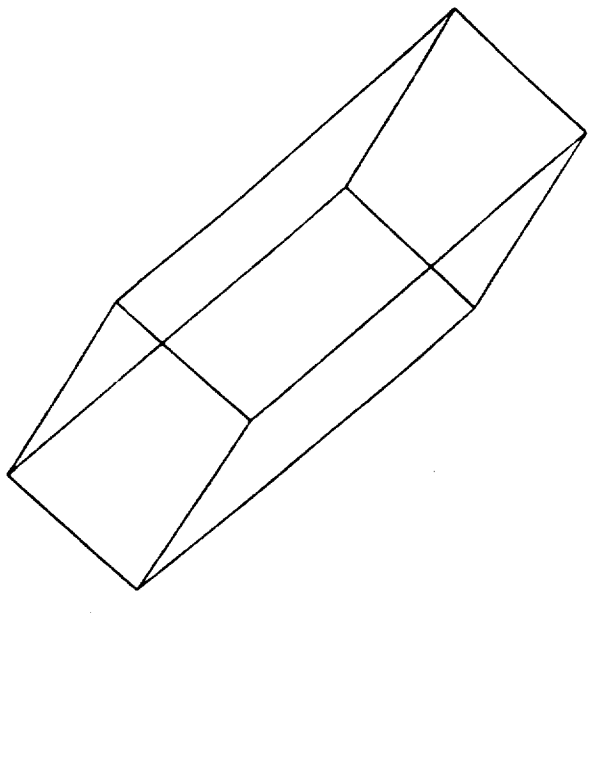

Figure 7. - Display From Program DRAW

6.0 SYSTEM LIBRARY PROCESSOR REFERENCE GUIDE

Sections 4 and 5 introduce commands of the SYSTEM LIBRARY PROCESSOR used for specific examples. The purpose of this section is to describe each command detailing all options and any restrictions. These command descriptions are application-independent as the intention is to provide a reference guide for all possible uses of the command.

All commands are described as they apply to ATTRIBUTE relations (section 3.1). Unless specifically stated otherwise, for any descriptions or prompts, the word attribute may be replaced by the word parameter to render the command description useful for PARAMETER relations.

All prompts appear as presented by the SYSTEM LIBRARY PROCESSOR with the exception of words enclosed by percent signs (% %). Such words represent words or phrases that depend upon the particular application and previous, user responses. All user responses are underlined and followed by a carriage return character (<CR>).

Before executing the SYSTEM LIBRARY PROCESSOR, the SYSTEM LIBRARY Database must reside in the current directory. To create the SYSTEM LIBRARY database and execute the SYSTEM LIBRARY PROCESSOR, see Section 4.1.

The initial prompt:

```
ENTER NUMBER OF CHARACTER/WORD FOR TARGET SYSTEM:
      4 FOR RIM ON PRIME/VAX
     10 FOR RIM ON NOS
      2 FOR SDRC/PRL
%NUMBER OF CHARACTERS PER WORD%<CR>
```

establishes a conversion factor used by the "BRV" and "BF" commands to retrieve/store character data from/to the database. The proper response depends upon the host computer system and/or the database management system used in execution of the REVIEWER and database read/write subroutines (generated by the FORMATTER).

The SYSTEM LIBRARY command menu follows:

```
SELECT OPTION:
BR  - TO BUILD RELATIONS
AR  - TO ADD TO AN EXISTING RELATION
BT  - TO BUILD A TEMPLATE
BF  - TO BUILD FORMATTER ROUTINES
BS  - TO BUILD A SCHEMA DUMP FOR DATA BASE
BRV - TO BUILD A REVIEWER INPUT FILE
BP  - TO BUILD PROGRAM DESCRIPTION
PT  - TO PRINT A TEMPLATE
LR  - TO LIST RELATIONS
PR  - TO PRINT A RELATION
X   - TO EXIT
```

6.1 BR - To Build Relations

The first step in constructing the SYSTEM LIBRARY is to define the relations. The "BR" command can be used at any time to add a relation to the SYSTEM LIBRARY.

```
BR<CR>

ENTER RELATION NAME (OR <CR> WHEN DONE):
%RELATION NAME%<CR>
```

Relation names are 1 to 8 alphanumeric characters and must be unique.

```
ENTER 1 FOR PARAMETER TYPE
   OR 2 FOR ATTRIBUTE TYPE
%OPTION%<CR>
```

A relation must be defined as PARAMETER or ATTRIBUTE type.

```
ENTER RELATION DESCRIPTION (80 CHARACTERS):
%RELATION DESCRIPTION%<CR>
```

The relation description is presented by the REVIEWER.

```
ENTER YOUR NAME (20 CHARACTERS):
%INTEGRATOR'S NAME%<CR>
```

This entry exists only to record the name of the program integrator responsible for creating this relation. Following the prompt:

```
DEFINE THE RELATION %RELATION NAME%:
```

loop begins to define each PARAMETER/ATTRIBUTE. The same information is required for both.

```
ENTER ATTRIBUTE NAME (OR <CR> WHEN DONE)>
%ATTRIBUTE NAME%<CR>
```

Names are 1 to 8 characters and may not end with the underscore (_) character. These names are also used as FORTRAN variable names in the FORMATTER, database read/write routines.

```
ENTER ATTRIBUTE TYPE>
%I,R,OR C%<CR>
```

PARAMETER/ATTRIBUTES may be typed integer (I), real (R), or character (C). If character type is selected,

```
ENTER NUMBER OF CHARACTERS IN STRING>
%STRING LENGTH%<CR>
```

the number of characters is solicited.

```
ENTER ATTRIBUTE DIMENSION (UP TO 3 SEPARATED BY COMMAS)>
%UP TO 3 DIMENSIONS%<CR>
```

PARAMETERS/ATTRIBUTES may be 1-, 2-, or 3-dimensional. A <CR> indicates a default single dimension of size one. For example, to dimension an attribute that is 2 x 3 x 6 enter:

```
2,3,6<CR>
```

The following report

```
DIMENSION = 2 X 3 X 6
```

indicates the selected dimension.

```
ENTER ATTRIBUTE DESCRIPTION>
%ATTRIBUTE DESCRIPTION%<CR>
ENTER UNIT OF MEASUREMENT (<CR> IF N/A)>
%UNIT%<CR>
```

Descriptions are limited to 80 characters. Units are a maximum of 16 characters. Both are presented by the REVIEWER.

```
ENTER OUTPUT FORMAT (<CR> IF N/A)>
<CR>
```

The intention of including an output format is to control output of the PARAMETER/ATTRIBUTE by the REVIEWER. However, this input is not currently recognized by the REVIEWER, and a <CR> response is sufficient.

If all inputs are correct, an affirmative response to:

```
OK TO ENTER THE ATTRIBUTE %ATTRIBUTE NAME% INTO THE DATA
BASE (Y/N) (<CR>=Y):
%Y OR N%<CR>
```

enters the PARAMETER/ATTRIBUTE into the SYSTEM LIBRARY PROCESSOR. A negative response disregards these entries.

```
ENTER ATTRIBUTE NAME (OR <CR> WHEN DONE)
<CR>
```

This completes the loop for defining a PARAMETER/ATTRIBUTE. The loop continues to define subsequent PARAMETERS/ATTRIBUTES until a <CR> is entered for:

```
ENTER RELATION NAME (OR <CR> WHEN DONE)>
<CR>
```

6.2 AR - To Add To An Existing Relation

PARAMETERS/ATTRIBUTES are added to an existing relation by this command. All prompts and responses are the same as the "BR" command. PARAMETERS/ATTRIBUTES are added to the end of the defined relation. As a warning, note that templates that access all PARAMETERS/ATTRIBUTES for an expanded relation (0 option in

the BT command described in Section 6.3) will access these new PARAMETERS/ATTRIBUTES in subsequent executions of the BRV and BF commands.

6.3 BT - To Build A Template

After relations are defined templates are designed to specify input/output from/to the database for a given application.

```
BT<CR>
ENTER TEMPLATE NAME (OR <CR> WHEN DONE):
%TEMPLATE NAME%<CR>
```

Template names are 1 to 8 characters in length and must be unique.

```
ENTER 1 IF INPUT TEMPLATE
   OR 2 IF OUTPUT TEMPLATE
%OPTION%<CR>
```

Input templates describe data to be retrieved from the database. Output templates describe data to be stored into the database. This option determines the type of subroutine produced by the BF command (GETDATA__ or PUTDATA__ described in Section 6.4).

Following the prompt:

IDENTIFY ALL %INPUT OR OUTPUT% RELATIONS FOR THE TEMPLATE

loop begins to describe how the template accesses a relation.

```
ENTER RELATION NAME (OR <CR> WHEN DONE):
%RELATION NAME%<CR>
ENTER 1 FOR PARAMETER TYPE
   OR 2 FOR ATTRIBUTE TYPE
%OPTION%<CR>
```

The relation name and type are entered. If the relation does not exist, the BR command is entered automatically to define a new relation. If the relation exists, an option allows review of the relation schema:

```
THE RELATION %RELATION NAME% IS ALREADY DEFINED AND HAS
%NUMBER OF ATTRIBUTES% ATTRIBUTES.
DO YOU WANT TO SEE THEM (Y/N) (<CR>=N):
%Y OR N%<CR>
```

An affirmative response displays the relation schema similar to the following sample.

```
ATTRIBUTE NAME  DIM1   DIM2   DIM3  TYPE  NUMBER OF CHARACTERS

A1                2      3      6  CHAR        20
A2                                   INT
A3               10                  REAL
```

A negative response results in no such display.

```
ENTER 0 IF ALL ATTRIBUTES ARE TO BE REPLACED
   OR 1 TO READ ATTRIBUTES TO BE REPLACED  FROM A FILE
   OR 2 TO ENTER ATTRIBUTES TO BE REPLACED FROM TERMINAL
%OPTION%<CR>
```

All or some of the PARAMETERS/ATTRIBUTES of a relation may be accessed. A "0" response indicates all PARAMETERS/ATTRIBUTES are to be accessed. A "1" response results in a query for a file name containing the desired list of names (one name per record). A "2" response results in the prompt:

```
ENTER ATTRIBUTE NAMES EACH FOLLOWED BY <CR>:
%ATTRIBUTE NAME%<CR> WHEN DONE
```

Names are then entered one per line. If a name is entered that does not exist in the relation, it is ignored and the prompt:

```
THE ATTRIBUTE NAME %ATTRIBUTE NAME% IS NOT DEFINED FOR
THIS RELATION.
TRY AGAIN:
```

allows input to continue with a warning. Again the option:

```
THE RELATION %RELATION NAME% IS ALREADY DEFINED AND HAS
%NUMBER OF ATTRIBUTES% ATTIRBUTES.
DO YOU WANT TO SEE THEM (Y/N) (<CR>=N):
```

appears to provide review of the relation schema. Regardless of the option taken, the prompt:

```
NEXT ATTRIBUTE:
```

indicates that name input is to continue.

After all desired names are entered, a <CR> ends the naming process and results in the report:

```
YOU HAVE IDENTIFIED %NUMBER OF ATTRIBUTES% ATTRIBUTES TO
BE RETRIEVED
```

For ATTRIBUTE type relations only, rows may be retrieved/replaced under conditional clauses and sorted by one or more attributes.

```
ENTER NUMBER OF CONDITIONS IN THE "WHERE" CLAUSE
   OR 0 FOR NO CONDITIONS
%NUMBER OF WHERE CONDITIONS%<CR>
```

Entering an integer greater than "0" will require the specification of WHERE clauses:

```
FOR CONDITION %CONDITION NUMBER% ENTER:
ATTRIBUTE NAME>
%ATTRIBUTE NAME%<CR>
LOGICAL OPERATOR>
%LOGICAL OPERATOR%<CR>
VALUE(%ATTRIBUTE TYPE%)>
%VALUE%<CR>
```

Attribute names must be defined for the relation. Logical operators are EQ, NE, LT, GT, LE, OR GE. Values must be of the same type as the attribute named.

If more than one WHERE condition is specified, the above sequence is repeated for each condition, and repetitions are separated by the query:

```
BOOLEAN OPERATOR FOR CONDITIONS %PREVIOUS CONDITION% AND
%NEXT CONDITION%>
%BOOLEAN OPERATOR%<CR>.
```

Legal choices for the boolean operator are AND or OR.

```
ENTER NUMBER OF CONDITIONS IN THE "SORT" CLAUSE
   OR 0 FOR NO CONDITIONS
%NUMBER OF SORT CONDITIONS%<CR>
```

Entering an integer greater than "0" requires the specification of sort conditions. For each sort condition, an attribute name:

```
FOR CONDITION %CONDITION NUMBER% ENTER:
ATTRIBUTE NAME>
%ATTRIBUTE NAME%<CR>]
```

and sort type:

```
1 FOR ASCENDING SORT OR
2 FOR DESCENDING SORT
%OPTION%<CR>
```

are entered. Sorting is performed by the specified sort order on the 1st attribute followed by the 2nd, 3rd...nth sort attribute.

For ATTRIBUTE relations referenced in output templates only, a replacement code is required:

```
ENTER REPLACEMENT CODE:
1 - ADD ALL TUPLES AT ONCE
2 - REPLACE ALL TUPLES AT ONCE
3 - REPLACE ALL TUPLES EXCLUSIVELY AT ONCE
4 - ADD TUPLES ONE AT A TIME
5 - REPLACE TUPLES ONE AT A TIME
6 - REPLACE TUPLES EXCLUSIVELY ONE AT A TIME
%REPLACEMENT CODE NUMBER%<CR>
```

This replacement code affects only the BF command and the rows are replaced in a relation as determined by following definitions.

1 - Rows are passed through the PUTDATA_ subroutine as arrays and are added to the end of the current relation.

2 - Rows are passed through the PUTDATA_ subroutine as arrays and replace the rows in the relation if they exist. If more rows are to be added than currently exists, new rows are added to the end of the current relation.

3 - The same as 2 except if the number of rows to be replaced is less than the number of rows in the current relation, the extra existing rows are deleted.

4, 5, and 6 - the same as 1, 2, and 3 respectively, except rows are passed in one at a time. Each call to a FORMATTER subroutine replaces one row.

```
OK TO ENTER THE RELATION %RELATION NAME% INTO THE
   TEMPLATE
%RELATION NAME% (Y/N) (<CR>=Y):
%Y OR N%<CR>
```

An affirmative response stores the relation into the template within the SYSTEM LIBRARY. If mistakes have been made in the relation specification, a negative response disregards all information for this relation from the template.

Here the loop for describing access to a relation ends with:

ENTER RELATION NAME (OR <CR> WHEN DONE):

At this point, another relation may be optionally specified for the template. When all relations are specified, a <CR> terminates the template definition.

ENTER TEMPLATE NAME (OR <CR> WHEN DONE):

Another template may be defined by entering a new template name. A <CR> terminates the BT command.

6.4 BF - To Build FORMATTER Routines

The term FORMATTER refers to conversion between database format and program internal format (program variables). An input FORMATTER routine (default name GETDATA_) retrieves data from the database to store in program variables. An output FORMATTER routine (default name PUTDATA__) retrieves data from program variables to store in the database. The information obtained from the input and output templates determines the content and structure of the FORMATTER GETDATA_ and PUTDATA_ routines, respectively.

```
BF<CR>
ENTER TEMPLATE NAME
OR <CR> FOR ALL RELATIONS
OR "Q" TO QUIT
%TEMPLATE NAME%<CR>
ENTER 1 IF INPUT TEMPLATE
   OR 2 IF OUTPUT TEMPLATE
%OPTION%<CR>
```

```
ENTER %GETDATA OR PUTDATA% FILE NAME
OR "Q" TO RETURN WITHOUT WRITING A FILE
%FILE NAME%<CR>
```

The form of the file name prompt depends upon whether the template is of type input or output. The file name entered may be any legal file name, including the complete directory path if needed. If the file exists, options are provided to overwrite the file or enter a new name.

```
Examples:BOXIN.FOR
         TOAIDE:[EXAMPLE.SOURCE]BOXIN.FOR
```

In most database systems, attribute names must be unique within a relation but can be duplicated in other relations. As the attribute names are used as FORTRAN variables in the FORMATTER routines, they must be unique for a selected template. For each set of duplicate attribute names within the template, the following prompts appear allowing new unique names to be entered:

```
THE FOLLOWING RELATIONS CONTAIN DUPLICATE
ATTRIBUTE/PARAMETER NAMES:
NUMBER     RELATION                NAME

       1   %RELATION NAME%         %ATTRIBUTE NAME%
       2   %RELATION NAME%         %ATTRIBUTE NAME%
ENTER A UNIQUE FORTRAN VARIABLE NAME FOR EACH
OF THE ABOVE DUPLICATES (8 CHARACTERS MAXIMUM):
       1   %RELATION NAME%         %ATTRIBUTE NAME%
%NEW NAME%<CR>
       2   %RELATION NAME%         %ATTRIBUTE NAME%
           %NEW NAME%<CR>
OK TO ENTER NEW NAMES IN THE TEMPLATE? (Y/N)
           %Y OR N%<CR>
```

Attribute names may be up to 8 characters in length. As the character length for ANSI standard variables is only 6 characters, an option is available to rename any attribute names of greater than 6 characters.

```
DO YOU REQUIRE ANSI STANDARD CODE? (Y/N) (<CR>=N)
%Y OR N%<CR>
```

A negative response maintains existing attribute names. An affirmative response results in the following prompts allowing each attribute name greater than 6 characters to be renamed.

```
THE FOLLOWING NAMES ARE GREATER THAN 6 CHARACTERS
AND CANNOT REPRESENT AN ANSI STANDARD VARIABLE NAME
ENTER A NEW NAME (6 CHARACTER MAXIMUM):
NUMBER    RELATION                NAME

   1      %RELATION NAME%         %ATTRIBUTE NAME%
%NEW NAME%<CR>
```

Variable names within programs may not match attribute names. Although the recommended approach is to maintain consistency between attribute names and FORMATTER variable names wherever possible, the variable names can be changed to match the program internal names.

```
DO YOU WANT TO RENAME ANY OTHER PARAMETERS/ATTRIBUTES?
(Y/N)(<CR>=N)
%Y OR N%<CR>
```

A negative response maintains existing attribute names. An affirmative response results in the following prompts allowing attributes to be renamed as FORMATTER variables.

```
ENTER CURRENT NAME OF PARAMETER/ATTRIBUTE TO CHANGE
OR "Q" TO QUIT
%ATTRIBUTE NAME%<CR>
%NEW NAME%<CR>
ENTER CURRENT NAME OF PARAMETER/ATTRIBUTE TO CHANGE
OR "Q" TO QUIT
Q<CR>
```

Attribute names within the database relations are never changed during any renaming process. Comments in the FORMATTER routine code link attribute names with the new FORTRAN variable names. Subsequent executions of the BF command for the same template uses these same new names without repeating the renaming process.

The number of rows in a relation is virtually unlimited. However, as attributes are stored in the FORMATTER routines as arrays, a dimension must be supplied for each attribute relation only (does not apply to parameters).

```
ENTER MAXIMUM NUMBER OF TUPLES TO BE REPLACED  FOR
RELATION %RELATION%
%PROGRAM'S DIMENSION%<CR>
```

This information is not stored and must be reentered with each execution of the "BF" command for a given template.

6.5 BS - To Build A Schema Dump For Database

Through the BR and AR commands, descriptions of the database schema are defined and stored in the SYSTEM LIBRARY. Execution of the BS command produces a file containing the database schema formatted for use in constructing the database.

```
BS<CR>
ENTER 1 FOR RIM SCHEMA
   OR 2 FOR SDRC/PRL SCHEMA
%OPTION%<CR>
```

Option "1" produces a RIM command input file which must be read by interactive RIM (the INPUT command) to create the database schema. Option "2" produces FORTRAN code which must be compiled and linked with a special RIM to PEARL conversion library and the PEARL FORTRAN library to produce the database schema.

```
ENTER SCHEMA DUMP FILE NAME
OR "Q" TO RETURN WITHOUT WRITING A FILE
%FILE NAME%<CR>
```

Regardless of the selected option, information must be written to a file, thus a file name must be supplied. If option "1" was selected, the name must be no more than 6 characters with an extension of .DAT. If option "2" was selected, the filename should have the extension .FOR. Directory location can precede the name.

Examples:

Option 1:
```
SCHEMA.DAT
TOAIDE:[EXAMPLE.CFG.DEFAULT]SCHEMA.DAT
```

Option 2:
```
PRLSCH.FOR
TOAIDE:[EXAMPLE.CFG.DEFAULT]PRLSCH.FOR
```

If the file exists, options are provided to overwrite the file or enter a new name.

```
ENTER RELATION NAME FOR SCHEMA DUMP
   OR <CR> FOR COMPLETE DATABASE SCHEMA DUMP
 OR "Q" TO QUIT
<CR>
```

The schema for a single relation is dumped by entering a pre-defined relation name. The recommended approach is to dump the complete schema and create the complete database via one file.

6.6 BRV - To Build A REVIEWER Input File

The REVIEWER is a generic program for database review and modification (Section 7.0). Although data are presented in two standard formats (parameter and attribute), choice of data subsets for review is specified through the template. The REVIEWER input file represents the means of transferring the

template information from the SYSTEM LIBRARY to the REVIEWER.

```
BRV<CR>
ENTER DIRECTORY LOCATION FOR REVIEWER INPUT FILES
OR <CR> TO CREATE A LOCAL REVIEWER FILE
%DIRECTORY LOCATION%<CR>
```

When using the recommended EASIE directory structure [3] the REVIEWER input file is transferred to the proper location by entering:

```
TOAIDE:[%SYSTEMID%.PROG]
```

If not, a <CR> produces a local file. In either case, the name of the file is:

```
%TEMPLATE NAME%.REV

ENTER TEMPLATE NAME
   OR <CR> TO CREATE REVIEWER FILES FOR ALL TEMPLATES
   OR "Q" TO QUIT
%TEMPLATE NAME%<C>.
```

REVIEWER input files may be constructed for a single template by entering the template name or for all defined templates by entering a <CR>.

```
ROW MODIFICATION FLAG FOR RELATION RELATION OF TEMPLATE
RELATION:
0 - FULL MODIFICATION
1 - MODIFY BUT CANNOT ADD ROWS
2 - CANNOT MODIFY ROWS
%OPTION%<CR>
```

REVIEWER input files contain flags for modifying attribute type data only that instruct the REVIEWER to modify as follows:

0 - Allow modification of any attributes in any row including adding or deleting rows.
1 - Allow modification of any attributes in any row but do not allow rows to be added or deleted.
2 - Allow only data review; no modification.

While the flags are written, this capability is not implemented in the current REVIEWER. Regardless of the selected option, the REVIEWER acts as if the flag equals "0". These flags are not

saved in the SYSTEM LIBRARY and must be reentered with each execution of BRV.

6.7 - BP - To Build Program Description

The BP command is not required for integration/interfacing of programs into a database system. However, if the completed application program is to be recognized and executed by the EASIE executive [4], this command must be executed for every application.

More than one FORTRAN or DCL program may be identified and executed as a single EASIE application. For this command, an application is referenced as a procedure and a program as a module. A procedure then consists of one or more modules to be executed sequentually. This is not to be confused with the EASIE procedure [4] which consists of one or more EASIE commands.

```
BP<CR>
ENTER NUMBER OF MODULES TO EXECUTE FOR THIS PROCEDURE >
%NUMBER OF MODULES%<CR>
```

For example, if a single program is to be executed as an EASIE application and is integrated with the database, the response is "1".

If the same program is interfaced, producing a preprocessor to read the database and format an input file, and a post processor to read an output file and store data into the database, the response is "3".

```
ENTER PROCEDURE NAME >
%PROCEDURE NAME%<CR>
```

The procedure name is the name used for all EASIE application commands.

Following this entry, a loop begins presenting a sequence of prompts/responses to describe each module of the procedure:

```
FOR MODULE # 1 ENTER :
NAME >
%MODULE NAME%<CR>
```

The module name is the file name of the FORTRAN executable file or the DCL command procedure for this module.

```
LOCATION (FULL DIRECTORY STRUCTURE OR
<CR> IF SAME AS LAST MODULE) >
%DIRECTORY LOCATION%<CR>
```

The module may be located anywhere on the disk system.

```
Example: TOAIDE:[%SYSTEMID%.PROG]
         DUA0:[TESTPROG]
         <CR> (If the program is in the current directory.)
```

Names should include the complete directory path including logical disk name.

```
ENTER 1 FOR FORTRAN PROGRAM
   OR 2 FOR COMMAND LANGUAGE PROCEDURE >
%OPTION%<CR>
```

The module may be either a FORTRAN program or a DCL command procedure.

```
ENTER INPUT TEMPLATE NAME >
%INPUT TEMPLATE NAME%<CR>
ENTER OUTPUT TEMPLATE NAME >
%OUTPUT TEMPLATE NAME%<CR>
```

Input and output template names are entered if they exist. If not applicable, a <CR> is entered.

```
IS A CONFIGURATION REQUIRED? (Y/N <CR> = Y) >
%Y OR N%<CR>
```

If the module interacts with a database or requires any type of input/output other than from/to the interactive terminal, the response is Y or <CR>. Otherwise the response is N.

```
ENTER NAME OF RESPONSIBLE PERSON OR
<CR> IF SAME AS LAST MODULE >
%INTEGRATOR'S NAME%<CR>
```

This entry exists only to record the name of the program integrator.

This completes the loop of module description. The loop is repeated for each module in the procedure. Following the last module description is the prompt:

```
OK TO ENTER PROCEDURE INTO PROGRAM DICTNRY? (Y/N <CR>=Y)
%Y OR N%<CR>
```

If all entries are correct, the response is Y. If not, the response is N and all module descriptions for the procedure are discarded.

```
OK FOR NEW PROCEDURE? (Y/N <CR>=Y) >
%Y OR N%<CR>
```

After the procedure is completely described, the opportunity to describe another procedure is presented. An affirmative response repeats the procedure description process. A negative response ends the procedure description process.

The procedure descriptions are stored in the SYSTEM LIBRARY. For this information to be available to the EASIE executive, a file, PROGRAM.PGD must be written to the PROGUFD area. (See WORKSPACE, EASIE Volume III.)

```
OK TO OUTPUT PROGRAM DATA TO .PROG AREA? (Y/N <CR>=Y) >
%Y OR N%<CR>
```

A negative response will end the BP command. An affirmative response will result in the prompt:

```
ENTER DIRECTORY LOCATION FOR PROGRAM DESCRIPTION FILES
OR <CR> TO CREATE A LOCAL PROGRAM DESCRIPTION FILE
%DIRECTORY LOCATION%<CR>
```

This entry must be the name of the PROGUFD directory used in the system workspace.

6.8 PT - To Print A Template

The PT command may be used at any time to print a template.

6.9 LR - To List Relations

The LR command may be used at any time to list all relations.

6.10 PR - To Print A Relation

The PR command may be used at any time to print a relation.

7.0 REVIEWER REFERENCE GUIDE

The REVIEWER is a generic program for reading, displaying, modifying, and retrieving/storing selected data from/into a database. The selection of data is accomplished via the REVIEWER input file created by the BRV command. The REVIEWER is designed to be executed from within the EASIE executive [4] but may be executed independently. To execute the REVIEWER standalone, the database must be in the current directory. A file named AIDE%terminal name%.TRM is created in the directory above the current directory and may be deleted after execution of the REVIEWER.

Data are presented by the REVIEWER in categories. A category is a subset of a relation. A subset can be anyone of three options specified in the template. These options are: (1) some or all parameters; (2) some or all attributes; and (3) some or all rows for attribute relations. Figure 8 is an example of category display for parameter relations. Figure 9 is an example of category display for attribute relations.

Notice that for parameter relations, values, names, subscripts, descriptions, and units are available with each display. Character values, descriptions, and units may be truncated. For attribute relations, only values are directly displayed. Character values may be truncated. Complete values, names, subscripts, descriptions, units, and data types are available with the expand (X) command.

A command menu follows the data display for the current category. Once this menu is mastered, it is helpful to toggle (T command) the menu off allowing a larger data display on the

terminal screen. The REVIEWER commands are described in the following subsections. The commands are similar for parameter and attribute relations. Each command is described with differences pertaining to the relation type being identified as required.

7.1 MODIFY Command, M

Parameter use:

M n : modify value (n = line#,name(subscript),or line range)

Attribute use:

M r c : modify value (r = row# or range;
c = column#,name(subscript), or range)

The MODIFY command alters category data locally to the REVIEWER. The altered data may be optionally replaced in the database when changing categories or exiting the REVIEWER. Parameters are identified by names or line numbers. A line is a parameter or an element of a multi-dimensional parameter. Elements of attribute categories must be identified by both row number and either name or column (attribute) number. After entering a MODIFY command (including <CR>) the following prompt appears.

ENTER NEW VALUE:

The new value is then entered. Alternatively, the new value can be appended to the MODIFY command. Using this method, text strings with embedded blank characters must be enclosed in single quotes (').

Examples:

Modify the parameter, NAME, to NEW NAME where 1 is the line number of NAME:

```
M 1<CR>
ENTER NEW VALUE:
NEW NAME<CR>

M 1 'NEW NAME'<CR>
```

Modify the parameter NAME to NEWNAME:

```
M NAME<CR>
ENTER NEW VALUE:
NEWNAME<CR>

M NAME NEWNAME<CR>
```

Modify row 2 column 3 to 10:

```
M 2 3<CR>
ENTER NEW VALUE:
10<CR>

M 2 3 10 <CR>
```

Modify row 2 column name FACE(3) to 10:

```
M 2 FACE(3)<CR>
ENTER NEW VALUE:
10<CR>

M 2 FACE(3) 10<CR>
```

Line, row, or column numbers may be replaced by a range specification to modify more than one element with a single command. In a range specification, arguments are delimited by a comma(,). Arguments separated by a colon(:) indicate all numbers in the range (inclusively) are to be included.

Examples:

Modify parameter line numbers 3,9,5,6, and 7:

```
M 3,9,5:7<CR>
```

Modify rows 8,2,3,4,5, and 1, and columns 8,9,10,11, and 12:

```
M 8,2,3:5,1 8:12<CR>
```

7.2 CHANGE CATEGORY Command, C

C n : change category (n = id or name)

When the REVIEWER is executed, the initial display is of the first category specified in the template. The display is changed to a different category by the CHANGE CATEGORY command. The new category is specified by the category number or name. The name is the relation name in the template. The number is the order of the relation in the template. Numbers and names can be displayed with the LIST CATEGORIES (CAT) command.

Examples:

Change to category 7

C 7<CR>

Change to category FACES

C FACES<CR>

If data have been modified in the current category, an option is present to replace the data in the database:

OK TO REPLACE CHANGES? (Y) >

An affirmative response or <CR> replaces the data. A negative response discards all modifications for this category.

7.3 NEXT PAGE Command, N

N n : next page (n = + or - pages)

A fixed number of lines (default of 20) are displayed on the terminal screen (page). If the number of lines (or rows) in the current category exceeds this fixed number, subsequent rows are displayed via the NEXT PAGE command. The parameter represents a page number relative to the current page.

Examples:

Lines 1:20 are initially displayed.

N 1<CR>

Displays lines 21:40.

N 3<CR>

Displays lines 81:100.

N -2<CR>

Displays lines 41:60.

7.4 REPRINT PAGE Command, R

R : reprint page

The REPRINT PAGE command redisplays the current page of data reflecting data modifications. The display reflects modifications, but these are not stored in the database until the category is changed or the REVIEWER is exited.

7.5 LINES PER PAGE Command, L

L n :n line#'s per page

The default number of parameter lines or attribute rows is 20. This value is changed with the LINES PER PAGE command.

Example:

Display 10 lines per page.

L 10<CR>

7.6 EXPAND DISPLAY LENGTH Command, X

Parameter use:

X n : expand line# n

Attribute use:

X r c : expand row# n, column# c

Because of screen size limitations, some information is not available from the REVIEWER page display. For parameters, character values exceeding 16 characters, descriptions exceeding 16 characters, and units exceeding 10 characters are truncated. For attribute data, character values exceeding 16 characters are truncated. Descriptions and units are not displayed. This information along with parameter/attribute types, character length when character type, and dimensions, is available through the EXPAND DISPLAY LENGTH command.

Parameter Example:

Expand parameter line number 1:

X 1<CR>

results in the output:

```
NAME      ! TYPE !  CHARS  !  SUBSCRIPT    ! UNIT
NAME      ! CHAR !      80 !               !
DESCRIPTION:
MODEL NAME
VALUE:
TEST OF MAKGEO FOR FIGURE 8
```

Attribute Example:

Expand row 3 column 2:

X 3 2<CR>

results in the output:

```
NAME      ! TYPE !  CHARS  !  SUBSCRIPT    ! UNIT

FACE      ! INT  !         !          2     !
DESCRIPTION:
FACE CONNECTIVITY (NODE 1 TO 2 TO 3 TO 4 (OR TO 1 IF 4=0)
TO 1)
VALUE:
                5
```

Names may be used in place of line or column numbers.

7.7 SET COLUMNS Command, S

S : set columns to be displayed

The SET COLUMNS command pertains only to attribute categories. This command may be used in either of two modes. If an integer parameter follows the S (separated by a space) the display is shifted that number of columns. Positive integers shift to the right and negative integers shift to the left.

Example:

If the first six columns are displayed:

S 6<CR>

displays columns 7:12. Any positive integer greater than the number of remaining columns to the right results in a display of the last 6 columns. Any negative integer less than the number of remaining columns to the left results in a display of the first 6 columns.

If no integer follows the "S", the option:

```
DESIGNATE THE COLUMNS OF INTEREST
"*" INDICATES ALL COLUMNS
"," IS DELIMITER BETWEEN COLUMNS
":" INDICATES A RANGE OF COLUMNS
<CR> DESCRIBES NO COLUMNS
```

provides a review of the attributes in the current category. An asterisk(*) indicates all attributes are to be reviewed while commas and colons are used to express a range of attributes. (Section 7.1 provides an explanation of range.) A <CR> bypasses the attribute review.

Sample output from the attribute review follows:

COL	NAME	TYPE	CHAR	DIMENSION	UNIT	DESCRIPTION
1	X	REAL	0		M	X COORDINATE
2	Y	REAL	0		M	Y COORDINATE
3	Z	REAL	0		M	Z COORDINATE

Following the attribute review, the option

```
DESIGNATE THE COLUMNS TO DISPLAY
<CR> INDICATES ALL COLUMNS
"," IS DELIMITER BETWEEN COLUMNS
":" INDICATES A RANGE OF COLUMNS
```

provides a selection of columns for display. A range of columns is specified. Alternatively, a <CR> displays the first 6 columns.

7.8 END COMMAND, E

E : end and save mods

The END command replaces any modifications for the current category only and terminates execution of the REVIEWER.

7.9 QUIT Command, Q

Q : quit without saving mods

The QUIT command terminates the REVIEWER discarding any modifications made to the current category.

7.10 HELP command, H

H : help

The HELP command is not implemented at the present time.

7.11 LIST CATAGORIES Command, CAT

CAT : list categories

The LIST CATEGORIES command provides a map of category identification numbers (order of relations in the template) and category names for the current template to be used with the CHANGE CATEGORY (C) command.

Example:

```
CAT<CR>

DESIGNATE THE CATEGORIES OF INTEREST
"*" INDICATES ALL CATEGORIES
"," IS DELIMITER BETWEEN CATEGORIES
":" INDICATES A RANGE OF CATEGORIES
*<CR>

     ID  CATEGORY  CATEGORY DESCRIPTION
      1  MODEL     MODEL INFORMATION
      2  NODES     NODE POINT COORDINATES
      3  FACES     CONNECTIVITY OF NODES TO FORM FACES
```

7.12 DEFINE REVIEW SUBSET Command, SUB

SUB : define review subset

The DEFINE REVIEW SUBSET command capability is not implemented.

7.13 TOGGLE command, T

T : toggle menu

By default, the menu is displayed with each page. Entering the TOGGLE MENU command deletes the menu from subsequent pages. Entering the TOGGLE MENU command again restores the menu display.

7.14 Error Messages

Entering an illegal range for line numbers, rows or columns results in the output:

```
ERROR IN RANGE SPECIFICATION - TRY AGAIN:
NOT A VALID COMMAND, PLEASE MODIFY COMMAND
```

and the command is ignored.

Entering a range of line numbers or rows of different types for modification results in the output:

```
ERROR - ALL %PARAMETERS OR ATTRIBUTES% IN RANGE ARE NOT
OF THE SAME TYPE
NOT A VALID COMMAND, PLEASE MODIFY COMMAND
```

and the command is ignored.

Entering any other illegal command results in the output:

NOT A VALID COMMAND, PLEASE MODIFY COMMAND

and the command is ignored.

```
                              CATEGORY       1:     MODEL
MODEL INFORMATION
-------------------------------------------------------------------------------
 L  !   PRESENT VALUE  !   NAME    !  SUBSCRIPT   ! DESCRIPTION       ! UNITS
-------------------------------------------------------------------------------
  1 ! TEST OF MAKGEO F ! NAME      !              ! MODEL NAME         !
  2 !       20.0000000 ! ROTATION  !          1   ! MODEL X,Y,Z ROTA  ! DEGREES
  3 !       30.0000000 !           !          2   !                   !
  4 !       40.0000000 !           !          3   !                   !
-------------------------------------------------------------------------------

M n : modify value (n = line#,name(subscript),or line range)
C n : change category (n = id or name)
N n : next page (n = + or - pages)
R : reprint page, L n : n line#'s per page, X n : expand line# n
E : end and save mods, Q : quit without saving mods, H : help
CAT : list categories, SUB : define review subset, T : toggle menu
EDIT:
>
```

Figure 8. - REVIEWER DISPLAY OF PARAMETER DATA

```
                              CATEGORY       3:     FACES
CONNECTIVITY OF NODES TO FORM FACES
-----------------------------------------------------
NAME   !  FACE     !  FACE     !  FACE     !  FACE
INDEX  !       1   !       2   !       3   !       4
-----------------------------------------------------
COL    !       1   !       2   !       3   !       4
-----------------------------------------------------
ROW :
     1!        1   !        2   !        3   !        4
     2!        2   !        6   !        7   !        3
     3!        6   !        5   !        8   !        7
     4!        5   !        1   !        4   !        8
     5!        4   !        3   !        7   !        8
     6!        5   !        6   !        2   !        1
-----------------------------------------------------

M r c : modify value(r= row# or range; c= column#,name(subscript), or range)
C n : change category (n = id or name)
N n : next page (n = + or - pages)
R : reprint page, L n : n rows per page, X r c : expand row# n, column# c
S : set columns to be displayed
E : end and save mods, Q : quit without saving mods, H : help
CAT : list categories, SUB : define review subset, T : toggle menu
EDIT:
>
```

Figure 9. - REVIEWER Display Of Attribute Data

APPENDIX A
USE OF THE SYSTEM LIBRARY PROCESSOR FOR THE COMPLEX EXAMPLE

```
$RUNDICT<CR>
ENTER NUMBER OF CHARACTER/WORD FOR TARGET SYSTEM:
      4 FOR RIM ON PRIME/VAX
     10 FOR RIM ON NOS
      2 FOR SDRC/PRL
4<CR>

SELECT OPTION:
BR  - TO BUILD RELATIONS
AR  - TO ADD TO AN EXISTING RELATION
BT  - TO BUILD A TEMPLATE
BF  - TO BUILD FORMATTER ROUTINES
BS  - TO BUILD A SCHEMA DUMP FOR DATA BASE
BRV - TO BUILD A REVIEWER INPUT FILE
BP  - TO BUILD PROGRAM DESCRIPTION
PT  - TO PRINT A TEMPLATE
LR  - TO LIST RELATIONS
PR  - TO PRINT A RELATION
X   - TO EXIT
BR<CR>

ENTER RELATION NAME (OR <CR> WHEN DONE):
NODES<CR>

ENTER 1 FOR PARAMETER TYPE
   OR 2 FOR ATTRIBUTE TYPE
2<CR>

ENTER RELATION DESCRIPTION (80 CHARACTERS):
NODE POINT COORDINATES<CR>

ENTER YOUR NAME (20 CHARACTERS):
JOHN DOE<CR>

DEFINE THE RELATION NODES    :

ENTER ATTRIBUTE NAME (OR <CR> WHEN DONE)>
X<CR>
```

```
ENTER ATTRIBUTE TYPE>
R<CR>

ENTER ATTRIBUTE DIMENSION (UP TO 3 SEPARATED BY COMMAS)>
1<CR>

DIMENSION = 1
ENTER ATTRIBUTE DESCRIPTION>
X COORDINATE<CR>

ENTER UNIT OF MEASUREMENT (<CR> IF N/A)>
M<CR>

ENTER OUTPUT FORMAT (<CR> IF N/A)>
<CR>

OK TO ENTER THE ATTRIBUTE X        INTO THE DATA BASE (Y/N) (<CR>=N):
Y<CR>


ENTER ATTRIBUTE NAME (OR <CR> WHEN DONE)>
Y<CR>

ENTER ATTRIBUTE TYPE>
R<CR>

ENTER ATTRIBUTE DIMENSION (UP TO 3 SEPARATED BY COMMAS)>
1<CR>

DIMENSION = 1
ENTER ATTRIBUTE DESCRIPTION>
Y COORDINATE<CR>

ENTER UNIT OF MEASUREMENT (<CR> IF N/A)>
M<CR>

ENTER OUTPUT FORMAT (<CR> IF N/A)>
<CR>

OK TO ENTER THE ATTRIBUTE Y        INTO THE DATA BASE (Y/N) (<CR>=N):
Y<CR>


ENTER ATTRIBUTE NAME (OR <CR> WHEN DONE)>
Z<CR>

ENTER ATTRIBUTE TYPE>
R<CR>

ENTER ATTRIBUTE DIMENSION (UP TO 3 SEPARATED BY COMMAS)>
1<CR>
```

APPENDIX A

```
DIMENSION = 1
ENTER ATTRIBUTE DESCRIPTION>
Z COORDINATE<CR>

ENTER UNIT OF MEASUREMENT (<CR> IF N/A)>
M<CR>

ENTER OUTPUT FORMAT (<CR> IF N/A)>
<CR>

OK TO ENTER THE ATTRIBUTE Z          INTO THE DATA BASE (Y/N) (<CR>=N):
Y<CR>


ENTER ATTRIBUTE NAME (OR <CR> WHEN DONE)>
<CR>


ENTER RELATION NAME (OR <CR> WHEN DONE):
FACES<CR>

ENTER 1 FOR PARAMETER TYPE
   OR 2 FOR ATTRIBUTE TYPE
2<CR>

ENTER RELATION DESCRIPTION (80 CHARACTERS):
CONNECTIVITY OF NODES TO FORM FACES<CR>

DEFINE THE RELATION FACES    :


ENTER ATTRIBUTE NAME (OR <CR> WHEN DONE)>
FACE<CR>

ENTER ATTRIBUTE TYPE>
I<CR>

ENTER ATTRIBUTE DIMENSION (UP TO 3 SEPARATED BY COMMAS)>
4<CR>

DIMENSION = 4
ENTER ATTRIBUTE DESCRIPTION>
FACE CONNECTIVITY (NODE 1 TO 2 TO 3 TO 4 (OR TO 1 IF 4=0) TO 1)<CR>

ENTER UNIT OF MEASUREMENT (<CR> IF N/A)>
<CR>

ENTER OUTPUT FORMAT (<CR> IF N/A)>
<CR>
```

```
OK TO ENTER THE ATTRIBUTE FACE      INTO THE DATA BASE (Y/N) (<CR>=N):
Y<CR>


ENTER ATTRIBUTE NAME (OR <CR> WHEN DONE)>
<CR>

ENTER RELATION NAME (OR <CR> WHEN DONE):
MODEL<CR>

ENTER 1 FOR PARAMETER TYPE
   OR 2 FOR ATTRIBUTE TYPE
1<CR>

ENTER RELATION DESCRIPTION (80 CHARACTERS):
MODEL INFORMATION<CR>

DEFINE THE RELATION MODEL    :

ENTER PARAMETER NAME (OR <CR> WHEN DONE)>
NAME<CR>

ENTER PARAMETER TYPE>
C<CR>

ENTER NUMBER OF CHARACTERS IN STRING>
80<CR>

ENTER PARAMETER DIMENSION (UP TO 3 SEPARATED BY COMMAS)>
1<CR>

DIMENSION = 1
ENTER PARAMETER DESCRIPTION>
MODEL NAME<CR>

ENTER UNIT OF MEASUREMENT (<CR> IF N/A)>
<CR>

ENTER OUTPUT FORMAT (<CR> IF N/A)>
<CR>

OK TO ENTER THE PARAMETER NAME      INTO THE DATA BASE (Y/N) (<CR>=N):
Y<CR>


ENTER PARAMETER NAME (OR <CR> WHEN DONE)>
ROTATION<CR>

ENTER PARAMETER TYPE>
R<CR>
```

```
ENTER PARAMETER DIMENSION (UP TO 3 SEPARATED BY COMMAS)>
3<CR>

DIMENSION = 3
ENTER PARAMETER DESCRIPTION>
MODEL X,Y,Z ROTATIONS RESPECTIVELY<CR>

ENTER UNIT OF MEASUREMENT (<CR> IF N/A)>
DEGREES<CR>

ENTER OUTPUT FORMAT (<CR> IF N/A)>
<CR>

OK TO ENTER THE PARAMETER ROTATION INTO THE DATA BASE (Y/N) (<CR>=N):
Y<CR>

ENTER PARAMETER NAME (OR <CR> WHEN DONE)>
<CR>


ENTER RELATION NAME (OR <CR> WHEN DONE):
<CR>


SELECT OPTION:
BR  - TO BUILD RELATIONS
AR  - TO ADD TO AN EXISTING RELATION
BT  - TO BUILD A TEMPLATE
BF  - TO BUILD FORMATTER ROUTINES
BS  - TO BUILD A SCHEMA DUMP FOR DATA BASE
BRV - TO BUILD A REVIEWER INPUT FILE
BP  - TO BUILD PROGRAM DESCRIPTION
PT  - TO PRINT A TEMPLATE
LR  - TO LIST RELATIONS
PR  - TO PRINT A RELATION
X   - TO EXIT
BT<CR>


ENTER TEMPLATE NAME (OR <CR> WHEN DONE):
MAKGEOIN<CR>

ENTER 1 IF INPUT TEMPLATE
   OR 2 IF OUTPUT TEMPLATE
1<CR>

IDENTIFY ALL INPUT  RELATIONS FOR THE TEMPLATE
```

```
ENTER RELATION NAME (OR <CR> WHEN DONE):
DIMEN<CR>

ENTER 1 FOR PARAMETER TYPE
   OR 2 FOR ATTRIBUTE TYPE
1<CR>

THE RELATION DIMEN    IS ALREADY DEFINED AND HAS       4 PARAMETERS.
DO YOU WANT TO SEE THEM (Y/N) (<CR>=N):
N<CR>

ENTER 0 IF ALL PARAMETERS ARE TO BE RETRIEVED
   OR 1 TO READ PARAMETERS TO BE RETRIEVED FROM A FILE
   OR 2 TO ENTER PARAMETERS TO BE RETRIEVED FROM THE TERMINAL
2<CR>

ENTER PARAMETER NAMES EACH FOLLOWED BY <CR>:
(PARAMETER NAME=<CR> WHEN DONE)
LENGTH<CR>
WIDTH<CR>
HEIGHT<CR>
<CR>

YOU HAVE IDENTIFIED       3 PARAMETERS TO BE RETRIEVED
OK TO ENTER THE RELATION DIMEN    INTO THE TEMPLATE MAKGEOIN (Y/N) (<CR>=N):
Y<CR>


ENTER RELATION NAME (OR <CR> WHEN DONE):
<CR>

ENTER TEMPLATE NAME (OR <CR> WHEN DONE):
MAKGEOOT<CR>


ENTER 1 IF INPUT TEMPLATE
   OR 2 IF OUTPUT TEMPLATE
2<CR>

IDENTIFY ALL OUTPUT RELATIONS FOR THE TEMPLATE


ENTER RELATION NAME (OR <CR> WHEN DONE):
NODES<CR>

ENTER 1 FOR PARAMETER TYPE
   OR 2 FOR ATTRIBUTE TYPE
2<CR>
```

```
THE RELATION NODES     IS ALREADY DEFINED AND HAS        3 ATTRIBUTES.
DO YOU WANT TO SEE THEM (Y/N) (<CR>=N):
N<CR>

ENTER 0 IF ALL ATTRIBUTES ARE TO BE REPLACED
   OR 1 TO READ ATTRIBUTES TO BE REPLACED  FROM A FILE
   OR 2 TO ENTER ATTRIBUTES TO BE REPLACED  FROM THE TERMINAL
0<CR>

ENTER NUMBER OF CONDITIONS IN THE "WHERE" CLAUSE
   OR 0 FOR NO CONDITIONS
0<CR>

ENTER NUMBER OF CONDITIONS IN THE "SORT" CLAUSE
   OR 0 FOR NO CONDITIONS
0<CR>

ENTER REPLACEMENT CODE:
1 - ADD ALL TUPLES AT ONCE
2 - REPLACE ALL TUPLES AT ONCE
3 - REPLACE ALL TUPLES EXCLUSIVELY AT ONCE
4 - ADD TUPLES ONE AT A TIME
5 - REPLACE TUPLES ONE AT A TIME
6 - REPLACE TUPLES EXCLUSIVELY ONE AT A TIME
3<CR>

OK TO ENTER THE RELATION NODES     INTO THE TEMPLATE MAKGEOOT (Y/N) (<CR>=N):
Y<CR>


ENTER RELATION NAME (OR <CR> WHEN DONE):
FACES<CR>

ENTER 1 FOR PARAMETER TYPE
   OR 2 FOR ATTRIBUTE TYPE
2<CR>

THE RELATION FACES     IS ALREADY DEFINED AND HAS        1 ATTRIBUTES.
DO YOU WANT TO SEE THEM (Y/N) (<CR>=N):
N<CR>

ENTER 0 IF ALL ATTRIBUTES ARE TO BE REPLACED
   OR 1 TO READ ATTRIBUTES TO BE REPLACED  FROM A FILE
   OR 2 TO ENTER ATTRIBUTES TO BE REPLACED  FROM THE TERMINAL
0<CR>

ENTER NUMBER OF CONDITIONS IN THE "WHERE" CLAUSE
   OR 0 FOR NO CONDITIONS
0<CR>
```

```
ENTER NUMBER OF CONDITIONS IN THE "SORT" CLAUSE
   OR 0 FOR NO CONDITIONS
0<CR>

ENTER REPLACEMENT CODE:
1 - ADD ALL TUPLES AT ONCE
2 - REPLACE ALL TUPLES AT ONCE
3 - REPLACE ALL TUPLES EXCLUSIVELY AT ONCE
4 - ADD TUPLES ONE AT A TIME
5 - REPLACE TUPLES ONE AT A TIME
6 - REPLACE TUPLES EXCLUSIVELY ONE AT A TIME
3<CR>

OK TO ENTER THE RELATION FACES     INTO THE TEMPLATE MAKGEOOT (Y/N) (<CR>=N):
Y<CR>

ENTER RELATION NAME (OR <CR> WHEN DONE):
<CR>

ENTER TEMPLATE NAME (OR <CR> WHEN DONE):
DRAWIN<CR>

ENTER 1 IF INPUT TEMPLATE
   OR 2 IF OUTPUT TEMPLATE
1<CR>

IDENTIFY ALL INPUT  RELATIONS FOR THE TEMPLATE


ENTER RELATION NAME (OR <CR> WHEN DONE):
MODEL<CR>

ENTER 1 FOR PARAMETER TYPE
   OR 2 FOR ATTRIBUTE TYPE
1<CR>

THE RELATION MODEL     IS ALREADY DEFINED AND HAS        2 PARAMETERS.
DO YOU WANT TO SEE THEM (Y/N) (<CR>=N):
N<CR>

ENTER 0 IF ALL PARAMETERS ARE TO BE RETRIEVED
   OR 1 TO READ PARAMETERS TO BE RETRIEVED FROM A FILE
   OR 2 TO ENTER PARAMETERS TO BE RETRIEVED FROM THE TERMINAL
0<CR>

OK TO ENTER THE RELATION MODEL     INTO THE TEMPLATE DRAWIN    (Y/N) (<CR>=N):
Y<CR>


ENTER RELATION NAME (OR <CR> WHEN DONE):
NODES<CR>
```

```
ENTER 1 FOR PARAMETER TYPE
   OR 2 FOR ATTRIBUTE TYPE
2<CR>

THE RELATION NODES     IS ALREADY DEFINED AND HAS        3 ATTRIBUTES.
DO YOU WANT TO SEE THEM (Y/N) (<CR>=N):
N<CR>

ENTER 0 IF ALL ATTRIBUTES ARE TO BE RETRIEVED
   OR 1 TO READ ATTRIBUTES TO BE RETRIEVED FROM A FILE
   OR 2 TO ENTER ATTRIBUTES TO BE RETRIEVED FROM THE TERMINAL
0<CR>

ENTER NUMBER OF CONDITIONS IN THE "WHERE" CLAUSE
   OR 0 FOR NO CONDITIONS
0<CR>

ENTER NUMBER OF CONDITIONS IN THE "SORT" CLAUSE
   OR 0 FOR NO CONDITIONS
0<CR>

OK TO ENTER THE RELATION NODES     INTO THE TEMPLATE DRAWIN    (Y/N) (<CR>=N):
Y<CR>


ENTER RELATION NAME (OR <CR> WHEN DONE):
FACES<CR>

ENTER 1 FOR PARAMETER TYPE
   OR 2 FOR ATTRIBUTE TYPE
2<CR>

THE RELATION FACES     IS ALREADY DEFINED AND HAS        1 ATTRIBUTES.
DO YOU WANT TO SEE THEM (Y/N) (<CR>=N):
N<CR>

ENTER 0 IF ALL ATTRIBUTES ARE TO BE RETRIEVED
   OR 1 TO READ ATTRIBUTES TO BE RETRIEVED FROM A FILE
   OR 2 TO ENTER ATTRIBUTES TO BE RETRIEVED FROM THE TERMINAL
0<CR>

ENTER NUMBER OF CONDITIONS IN THE "WHERE" CLAUSE
   OR 0 FOR NO CONDITIONS
0<CR>

ENTER NUMBER OF CONDITIONS IN THE "SORT" CLAUSE
   OR 0 FOR NO CONDITIONS
0<CR>

OK TO ENTER THE RELATION FACES     INTO THE TEMPLATE DRAWIN    (Y/N) (<CR>=N):
Y<CR>
```

```
ENTER RELATION NAME (OR <CR> WHEN DONE):
<CR>

ENTER TEMPLATE NAME (OR <CR> WHEN DONE):
<CR>


SELECT OPTION:
BR  - TO BUILD RELATIONS
AR  - TO ADD TO AN EXISTING RELATION
BT  - TO BUILD A TEMPLATE
BF  - TO BUILD FORMATTER ROUTINES
BS  - TO BUILD A SCHEMA DUMP FOR DATA BASE
BRV - TO BUILD A REVIEWER INPUT FILE
BP  - TO BUILD PROGRAM DESCRIPTION
PT  - TO PRINT A TEMPLATE
LR  - TO LIST RELATIONS
PR  - TO PRINT A RELATION
X   - TO EXIT
BS<CR>

ENTER 1 FOR RIM SCHEMA
   OR 2 FOR SDRC/PRL SCHEMA
1<CR>

ENTER SCHEMA DUMP FILE NAME
OR "Q" TO RETURN WITHOUT WRITING A FILE
SCHEMA.DAT<CR>

ENTER RELATION NAME FOR SCHEMA DUMP
   OR <CR> FOR COMPLETE DATABASE SCHEMA DUMP
   OR "Q" TO QUIT
<CR>


SELECT OPTION:
BR  - TO BUILD RELATIONS
AR  - TO ADD TO AN EXISTING RELATION
BT  - TO BUILD A TEMPLATE
BF  - TO BUILD FORMATTER ROUTINES
BS  - TO BUILD A SCHEMA DUMP FOR DATA BASE
BRV - TO BUILD A REVIEWER INPUT FILE
BP  - TO BUILD PROGRAM DESCRIPTION
PT  - TO PRINT A TEMPLATE
LR  - TO LIST RELATIONS
PR  - TO PRINT A RELATION
X   - TO EXIT
BF<CR>
```

```
ENTER TEMPLATE NAME
OR <CR> FOR ALL RELATIONS
OR "Q" TO QUIT
MAKGEOIN<CR>

ENTER GETDATA FILE NAME
OR "Q" TO RETURN WITHOUT WRITING A FILE
MAKGEOIN.FOR<CR>

DO YOU REQUIRE ANSI STANDARD CODE? (Y/N) (<CR>=N)
N<CR>

DO YOU WANT TO RENAME ANY OTHER PARAMETERS/ATTRIBUTES? (Y/N) (<CR>=N)
N<CR>


SELECT OPTION:
BR  - TO BUILD RELATIONS
AR  - TO ADD TO AN EXISTING RELATION
BT  - TO BUILD A TEMPLATE
BF  - TO BUILD FORMATTER ROUTINES
BS  - TO BUILD A SCHEMA DUMP FOR DATA BASE
BRV - TO BUILD A REVIEWER INPUT FILE
BP  - TO BUILD PROGRAM DESCRIPTION
PT  - TO PRINT A TEMPLATE
LR  - TO LIST RELATIONS
PR  - TO PRINT A RELATION
X   - TO EXIT
BF<CR>

ENTER TEMPLATE NAME
OR <CR> FOR ALL RELATIONS
OR "Q" TO QUIT
MAKGEOOT<CR>

ENTER PUTDATA FILE NAME
OR "Q" TO RETURN WITHOUT WRITING A FILE
MAKGEOOT.FOR<CR>

DO YOU REQUIRE ANSI STANDARD CODE? (Y/N) (<CR>=N)
N<CR>

DO YOU WANT TO RENAME ANY OTHER PARAMETERS/ATTRIBUTES? (Y/N) (<CR>=N)
N<CR>

ENTER MAXIMUM NUMBER OF TUPLES TO BE REPLACED  FOR RELATION NODES
8<CR>

ENTER MAXIMUM NUMBER OF TUPLES TO BE REPLACED  FOR RELATION FACES
6<CR>
```

```
SELECT OPTION:
BR  - TO BUILD RELATIONS
AR  - TO ADD TO AN EXISTING RELATION
BT  - TO BUILD A TEMPLATE
BF  - TO BUILD FORMATTER ROUTINES
BS  - TO BUILD A SCHEMA DUMP FOR DATA BASE
BRV - TO BUILD A REVIEWER INPUT FILE
BP  - TO BUILD PROGRAM DESCRIPTION
PT  - TO PRINT A TEMPLATE
LR  - TO LIST RELATIONS
PR  - TO PRINT A RELATION
X   - TO EXIT
BF<CR>

ENTER TEMPLATE NAME
OR <CR> FOR ALL RELATIONS
OR "Q" TO QUIT
DRAWIN<CR>

ENTER GETDATA FILE NAME
OR "Q" TO RETURN WITHOUT WRITING A FILE
DRAWIN.FOR<CR>

DO YOU REQUIRE ANSI STANDARD CODE? (Y/N) (<CR>=N)
N<CR>

DO YOU WANT TO RENAME ANY OTHER PARAMETERS/ATTRIBUTES? (Y/N) (<CR>=N)
N<CR>

ENTER MAXIMUM NUMBER OF TUPLES TO BE RETRIEVED FOR RELATION NODES
100<CR>

ENTER MAXIMUM NUMBER OF TUPLES TO BE RETRIEVED FOR RELATION FACES
100<CR>


SELECT OPTION:
BR  - TO BUILD RELATIONS
AR  - TO ADD TO AN EXISTING RELATION
BT  - TO BUILD A TEMPLATE
BF  - TO BUILD FORMATTER ROUTINES
BS  - TO BUILD A SCHEMA DUMP FOR DATA BASE
BRV - TO BUILD A REVIEWER INPUT FILE
BP  - TO BUILD PROGRAM DESCRIPTION
PT  - TO PRINT A TEMPLATE
LR  - TO LIST RELATIONS
PR  - TO PRINT A RELATION
X   - TO EXIT
BRV<CR>
```

APPENDIX B
SOURCE CODE FILES

THE PROGRAM MAKGEO AS CONTAINED IN THE FILE MAKGEO.FOR:

```
      PROGRAM MAKGEO
C
      COMMON /INDIMEN/ LENGTH,WIDTH,HEIGHT
      REAL LENGTH
      COMMON /OUTGEO/ X(8),Y(8),Z(8),NG,FACE(4,6),NF
      INTEGER FACE
C
      CALL GETDATA_
C
      NG = 8
      NF = 6
      HL = LENGTH/2.
      HW = WIDTH/2.
      HH = HEIGHT/2.
C
      X(1) = -HL
      Y(1) = -HH
      Z(1) = -HW
      X(2) =  HL
      Y(2) = -HH
      Z(2) = -HW
      X(3) =  HL
      Y(3) =  HH
      Z(3) = -HW
      X(4) = -HL
      Y(4) =  HH
      Z(4) = -HW
      X(5) = -HL
      Y(5) = -HH
      Z(5) =  HW
      X(6) =  HL
      Y(6) = -HH
      Z(6) =  HW
      X(7) =  HL
      Y(7) =  HH
      Z(7) =  HW
      X(8) = -HL
      Y(8) =  HH
      Z(8) =  HW
      FACE(1,1) = 1
      FACE(2,1) = 2
      FACE(3,1) = 3
      FACE(4,1) = 4
      FACE(1,2) = 2
      FACE(2,2) = 6
      FACE(3,2) = 7
      FACE(4,2) = 3
      FACE(1,3) = 6
      FACE(2,3) = 5
      FACE(3,3) = 8
```

C-2

```
      FACE(4,3) = 7
      FACE(1,4) = 5
      FACE(2,4) = 1
      FACE(3,4) = 4
      FACE(4,4) = 8
      FACE(1,5) = 4
      FACE(2,5) = 3
      FACE(3,5) = 7
      FACE(4,5) = 8
      FACE(1,6) = 5
      FACE(2,6) = 6
      FACE(3,6) = 2
      FACE(4,6) = 1
C
      CALL PUTDATA
C
      END
```

APPENDIX B

THE SUBROUTINE GETDATA_ FOR USE WITHIN MAKGEOIN.FOR

```
      SUBROUTINE GETDATA_
C
      COMMON /DBNAME_/ DBASE_,DBOPN_
      REAL*8 DBASE_
      LOGICAL DBOPN_
C
C*** THE PARAMETERS FOR THE RELATION DIMEN    ARE:
      REAL     LENGTH
      REAL     WIDTH
      REAL     HEIGHT
C
C   LOAD REQUIRED COMMON BLOCKS HERE:
C
        INCLUDE 'MAKGEOIN.COMMON/LIST'
        DBASE_ = (8HDATADB  )
C
        CALL R1001_(LENGTH,WIDTH,HEIGHT)
C
C   MAKE ANY NECESSARY RE-ASSIGNMENTS HERE:
C
C       INCLUDE 'MAKGEOIN.ASSIGN/LIST'
C
        RETURN
C
        END
```

APPENDIX B

THE SUBROUTINE PUTDATA_ FOR USE WITHIN MAKGEOOT.FOR

```
      SUBROUTINE PUTDATA_
C
      COMMON /DBNAME_/ DBASE_,DBOPN_
      REAL*8 DBASE_
      LOGICAL DBOPN_
C
C*** THE ATTRIBUTES FOR THE RELATION NODES    ARE:
      REAL     X       (      8)
      REAL     Y       (      8)
      REAL     Z       (      8)
C*** THE ATTRIBUTES FOR THE RELATION FACES    ARE:
      INTEGER  FACE    (      4,     6)
C
C   LOAD REQUIRED COMMON BLOCKS HERE:
C
      INCLUDE 'MAKGEOOT.COMMON/LIST'
      DBASE_ = (8HDATADB  )
C
C   MAKE ANY NECESSARY RE-ASSIGNMENTS HERE:
C
C     INCLUDE 'MAKGEOOT.ASSIGN/LIST'
C
      CALL W1001_(X,Y,Z,N001_)
      CALL W1002_(FACE,N002_)
C
      RETURN
C
      END
```

CONTENTS OF THE FILE MAKGEO.COM

```
$FOR MAKGEO.FOR
$FOR MAKGEOIN.FOR
$FOR MAKGEOOT.FOR
$LINK MAKGEO.OBJ -
     +MAKGEOIN.OBJ -
     +MAKGEOOT.OBJ -
     +LOADRIM/LIB
```

APPENDIX B

THE FILE DRAWIN.FOR MODIFIED AS NEEDED:

```
C      SUBROUTINE GETDATA_
C
       COMMON /DBNAME_/ DBASE_,DBOPN_
       REAL*8 DBASE
       LOGICAL DBOPN_
C
C*** THE PARAMETERS FOR THE RELATION MODEL    ARE:
       CHARACTER*     80 NAME
       REAL      ROTATION(      3)
C*** THE ATTRIBUTES FOR THE RELATION NODES    ARE:
       REAL      X         (    100)
       REAL      Y         (    100)
       REAL      Z         (    100)
C*** THE ATTRIBUTES FOR THE RELATION FACES    ARE:
       INTEGER   FACE      (      4,    100)
C
C    LOAD REQUIRED COMMON BLOCKS HERE:
C
C      INCLUDE 'DRAWIN.COMMON/LIST'
       DBASE_ = (8HDATADR   )
C
       CALL R1001_(NAME,ROTATION)
       CALL R1002_(X,Y,Z,N002_)
       CALL R1003_(FACE,N003_)
C
C    MAKE ANY NECESSARY RE-ASSIGNMENTS HERE:
C
       INCLUDE 'DRAWIN.ASSIGN/LIST'
C
C      RETURN
C
       END
```

APPENDIX B

CONTENTS OF THE FILE DRAWIN.ASSIGN:

```
C   CONVERT ROTATION FROM DEGREES TO RADIANS:
C
      DEGRAD = 3.141592654/180.
      DO 10 I=1,3
         ROTATION(I) = ROTATION(I) * DEGRAD
   10 CONTINUE
C
C   WRITE DATA TO INPUT FILE 'DRAW.DAT':
C
      OPEN(8,FILE='DRAW.DAT',STATUS='UNKNOWN')
      WRITE(8,'(A)') NAME
      WRITE(8,'(3E12.4)') (ROTATION(I),I=1,3)
      WRITE(8,'(I5)') NO02
      WRITE(8,'(3E12.4)') (X(I),Y(I),Z(I),I=1,NO02_)
      WRITE(8,'(I5)') NO03
      WRITE(8,'(4I5)') ((FACE(J,I),J=1,4),I=1,NO03_)
      CLOSE(8)
```

CONTENTS OF THE FILE DRAWIN.COM:

```
$FOR DRAWIN.FOR
$LINK DRAWIN.OBJ -
     +LOADRIM/LIB
```

APPENDIX C
TEMPLATE MAKGEOIN

TO ESTABLISH VALUES FOR LENGTH, WIDTH AND HEIGHT

```
$REVIEW MAKGEOIN<CR>

                              CATEGORY        1:     DIMEN
BOX DIMENSIONS

-------------------------------------------------------------------------------
 L  !   PRESENT VALUE  !   NAME    !  SUBSCRIPT   ! DESCRIPTION      ! UNITS
-------------------------------------------------------------------------------
  1 !        0.0000000 ! LENGTH    !              ! BOX LENGTH       ! M
  2 !        0.0000000 ! WIDTH     !              ! BOX WIDTH        ! M
  3 !        0.0000000 ! HEIGHT    !              ! BOX HEIGHT       ! M
-------------------------------------------------------------------------------

M n : modify value (n = line#,name(subscript),or line range)
C n : change category (n = id or name)
N n : next page (n = + or - pages)
R : reprint page, L n : n line#'s per page, X n : expand line# n
E : end and save mods, Q : quit without saving mods, H : help
CAT : list categories, SUB : define review subset, T : toggle menu

EDIT:
> M LENGTH 1<CR>
> M WIDTH 2<CR>
> M HEIGHT 3<CR>
> R<CR>

                              CATEGORY        1:     DIMEN
BOX DIMENSIONS

-------------------------------------------------------------------------------
 L  !   PRESENT VALUE  !   NAME    !  SUBSCRIPT   ! DESCRIPTION      ! UNITS
-------------------------------------------------------------------------------
  1 !        1.0000000 ! LENGTH    !              ! BOX LENGTH       ! M
  2 !        2.0000000 ! WIDTH     !              ! BOX WIDTH        ! M
  3 !        3.0000000 ! HEIGHT    !              ! BOX HEIGHT       ! M
-------------------------------------------------------------------------------

M n : modify value (n = line#,name(subscript),or line range)
C n : change category (n = id or name)
N n : next page (n = + or - pages)
R : reprint page, L n : n line#'s per page, X n : expand line# n
E : end and save mods, Q : quit without saving mods, H : help
CAT : list categories, SUB : define review subset, T : toggle menu

EDIT:
> E<CR>
```

APPENDIX D
TEMPLATE DRAWIN

TO ESTABLISH VALUES FOR NAME AND ROTATIONS

```
$REVIEW DRAWIN<CR>

                                CATEGORY       1:     MODEL
MODEL INFORMATION

-------------------------------------------------------------------------------
 L  !   PRESENT VALUE  !   NAME    !  SUBSCRIPT   ! DESCRIPTION       ! UNITS
-------------------------------------------------------------------------------
  1 !                  ! NAME      !              ! MODEL NAME        !
  2 !        0.0000000 ! ROTATION  !          1   ! MODEL X,Y,Z ROTA  ! DEGREES
  3 !        0.0000000 !           !          2   !                   !
  4 !        0.0000000 !           !          3   !                   !
-------------------------------------------------------------------------------

M n : modify value (n = line#,name(subscript),or line range)
C n : change category (n = id or name)
N n : next page (n = + or - pages)
R : reprint page, L n : n line#'s per page, X n : expand line# n
E : end and save mods, Q : quit without saving mods, H : help
CAT : list categories, SUB : define review subset, T : toggle menu
EDIT:
> M 1 'TEST BOX'<CR>
> M 2 20<CR>
> M 3 30<CR>
> M 4 40<CR>
> R<CR>

                                CATEGORY       1:     MODEL
MODEL INFORMATION

-------------------------------------------------------------------------------
 L  !   PRESENT VALUE  !   NAME    !  SUBSCRIPT   ! DESCRIPTION       ! UNITS
-------------------------------------------------------------------------------
  1 ! 'TEST BOX'       ! NAME      !              ! MODEL NAME        !
  2 !       20.0000000 ! ROTATION  !          1   ! MODEL X,Y,Z ROTA  ! DEGREES
  3 !       30.0000000 !           !          2   !                   !
  4 !       40.0000000 !           !          3   !                   !
-------------------------------------------------------------------------------

M n : modify value (n = line#,name(subscript),or line range)
C n : change category (n = id or name)
N n : next page (n = + or - pages)
R : reprint page, L n : n line#'s per page, X n : expand line# n
E : end and save mods, Q : quit without saving mods, H : help
CAT : list categories, SUB : define review subset, T : toggle menu
EDIT:
> E<CR>

$
```

APPENDIX E
BUILD_EASIE

Version 1.0 for VAX/VMS Running EASIE With RIM

(You should have a good knowledge of EASIE before attempting to use this program)

BUILD_EASIE is a utility program designed to aid in the installation of programs into the Environment for Application Software Integration and Execution (EASIE). When the construction of large numbers of relations would make the interactive processor RUNDICT tedious to use, BUILD_EASIE provides an easy-to-use alternative file interface. The user creates a data file called EASIE.DAT (described in the following section) which contains the relation and template data for one or more programs to be installed into EASIE. BUILD_EASIE provides commands that operate on EASIE.DAT and then prompts for the desired relations and templates to be created. The BUILD_EASIE program generates a VAX/VMS command procedure called EASIE.COM which includes the calls to RUNDICT and the answers to all of its prompts. The long series of prompts in RUNDICT need not be answered interactively and the database structure can be easily maintained in EASIE.DAT. Changing the database structure requires only simple editing of EASIE.DAT and a re-execution of BUILD_EASIE and EASIE.COM. Currently BUILD_EASIE supports only PARAMETER type relations.

EASIE.COM creates the relations, templates, FORMATTER routines (GETDATA_/PUTDATA_), the REVIEWER files, the database schema, and the three RIM database files. The FORMATTER routines are created in files with the template name and a .FOR

extension. REVIEWER files have the template name with a .REV extension. The data base schema is created in SCHEMA.DAT.

BUILD_EASIE also creates for each input template a command procedure called a reviewer loader. The reviewer loader calls the EASIE REVIEWER and loads values into the database for each variable contained in the template. Editing the reviewer loader command procedure supplies the values for the variables in the template. Reviewer loaders are created in files with the input template name and a .COM extension. These files provide a method of initializing all variables to values representative of some baseline check case.

The data file can be sorted, by variable name, by typing the sort command provided in BUILD_EASIE. The sorted data file is not used by BUILD_EASIE but is intended to be a reference tool.

BUILD_EASIE is run at the terminal by typing RUN BUILD_EASIE. The first prompt requires the EASIE data file name. A carriage return will assign the default file name of EASIE.DAT. The syntax of the three BUILD_EASIE commands is then displayed. The three commands are: (1) "R <relation name>" to build relations; (2) "T <template name>" to build templates; and, (3) "S" to sort EASIE.DAT. A simple carriage return will end the execution of BUILD_EASIE. EASIE.COM and any reviewer loaders will reside in the current directory. EASIE.COM can now be executed at the terminal or in batch mode.

APPENDIX E

Creating the EASIE Data File:

EASIE.DAT is a formatted file that contains the data for an unlimited number of relations and templates. Template names must begin with the letter O for output templates or the letter I for input templates. The format of EASIE.DAT is as follows:

```
* Relation_name   Relation_description
      Date  User_name
Variable_name  Data_type  dimensions  Units  Format  Description
      template_names...
Variable_name  Data_type  dimensions  Units  Format  Description
      template_names...
.
.
.
* Relation_name   Relation_description
      Date  User_name
Variable_name  Data_type  dimensions  Units  Format  Description
      template_names...
.
.
.
```

All data must appear in the proper columns for BUILD_EASIE to work properly. All variables listed below a relation name are placed in that relation until a new relation name is declared or the end of the file is reached. The template names refer to the variable listed on the line above them. All templates in which the variable will be included must be listed. No field in the data file may begin with the $ character. The column and field specifications for each piece of information is as follows:

APPENDIX E

EASIE Data Item	Comments	Beginning Column #
Relation name	* in column #2 must be present	4
Relation Description	40 characters maximum	13
Date	Date relation is created (not used by EASIE)	4
User Name	Name of programmer	13
Variable Name	7 characters maximum	2
Data Type	Must be R, C, D, or I	10
Dimensions	Up to 3 separated by commas with no embedded spaces. For character type the length must appear first, followed by a space then the dimensions	12
Units	Anything, use "+" for no units	25
Format	FORTRAN output format, or "+" for default	42
Variable Description	40 characters maximum	59
Template Names	Any number separated by blanks, begin with I or O	4

Example Run of BUILD_EASIE:

The following is a sample scenerio for installing a program into EASIE using BUILD_EASIE. Terminal input and output is shown in small type with the user inputs in bold face.

We begin by displaying our default directory which contains the BUILD_EASIE executable and the EASIE.DAT file.

$ DIR

Directory DISK:[EASIE]

BUILD_EASIE.EXE;1 EASIE.DAT;1

Total of 2 files.

APPENDIX E

$ TYPE EASIE.DAT

```
* REL_1    RELATION FOR TEST PROGRAM
  01/01/88 JOHN SMITH
XXX      R 1            FEET           +          DISTANCE TO SATELLITE
  ITEMP OTEMP
YYY      I 2,3,2        INCHES         I5         SIZE OF PARTS
  ITEMP OTEMP
ZZZ      R 5,6          METERS         +          COORDINATES
  OTEMP ITABLE
* REL_2    RELATION NO. 2
  03/03/88 JOE BLOW
DESCR    C 80 1         +              +          PROGRAM DESCRIPTION.
  OTEMP ITABLE
VOL      R 1            M**3           +          VOLUME OF OBJECT
  OTEMP
COUNT    I 1            +              I3         NUMBER OF PARTS
  ITEMP
```

This EASIE.DAT file contains the data for two relations and three templates. Relation REL_1 contains the variables XXX, YYY, and ZZZ. Relation REL_2 contains the variables DESCR, VOL, and COUNT. Input template ITEMP contains the variables XXX, YYY, DESCR, and COUNT. Output template OTEMP contains the variables XXX, YYY, ZZZ, DESCR, and VOL. Input template ITABLE contains the variables ZZZ and DESCR.

Now we run BUILD_EASIE to create the command procedures and the sorted data file. EASIE requires that a relation be created before a template can use it, so you must be sure to build the relations before building the templates.

APPENDIX E

```
$ RUN BUILD_EASIE

Build_Easie

VER. 1.0   VAX/VMS - RIM,  5-5-87

ENTER EASIE DATA FILE NAME, <CR> = EASIE.DAT
-> EASIE.DAT

ENTER: R <RELATION NAME>      TO BUILD RELATIONS
       T <TEMPLATE NAME>      TO BUILD TEMPLATES
       S                      TO SORT THE DATA FILE
       <CR>                   TO END

-> R REL_1
RELATION REL_1     COMPLETED
-> R REL_2
RELATION REL_2     COMPLETED
-> T ITEMP
TEMPLATE ITEMP     COMPLETED
-> T OTEMP
TEMPLATE OTEMP     COMPLETED
-> S
THE SORTED DATA IS IN FILE: SORT.DAT
->
FORTRAN STOP

$ DIR

Directory DISK:[EASIE]

BUILD_EASIE.EXE;1      EASIE.COM;1                    EASIE.DAT;1
ITEMP.COM;1
SORT.DAT;1

Total of 5 files.
```

We see that the command procedure EASIE.COM, the reviewer loader command procedure ITEMP.COM, and the sorted data file SORT.DAT have been created in our current directory. Notice that the template ITABLE is not listed in the directory because it was not specified to be built. EASIE.DAT may contain many relations and templates that are not referenced on any given run of BUILD_EASIE.

APPENDIX E

Since this is a new EASIE system, we must create the initial dictionary. We run BUILDDICT.COM to create the files DICT1.DAT, DICT2.DAT, and DICT3.DAT. The symbol RIM must also exist to execute RIM.

```
$ RIM == "RUN DISK1:[RIM.RIM5]RIM.EXE"
$ @TOAIDE:[BUILD_DICT]BUILDDICT.COM

BEGIN RIM ----- VAX VERSION 5.0   UD23          87/05/14    13.18.50


RIM COMMAND MODE
ENTER "MENU" FOR MENU MODE

R>

BEGIN RIM SCHEMA COMPILATION


RIM SCHEMA COMPILATION FOR DICT      IS COMPLETE

        EXISTING RELATIONS AS OF 87/05/14   13.19.15

                ATTRIB
                PARAM
                RELATION
                TEMPLATE
                TP.CNTRL
                TP.INDEX
                TP.SORT
                TP.WHERE
                TPC.INDX
                PROGRAM
                FORNAM

FORTRAN STOP

$ DIR

Directory DISK:[EASIE]

BUILD_EASIE.EXE;1   DICT1.DAT;1         DICT2.DAT;1         DICT3.DAT;1
EASIE.COM;1         EASIE.DAT;1         ITEMP.COM;1         SORT.DAT;1

Total of 8 files.
```

Next, the command file EASIE.COM is executed with the output written to RUNEASIE.OUT. The output can be very lengthy and is most often deleted. But the output file can be useful if problems develop when running EASIE.COM. With each subsequent execution of EASIE.COM, the dictionary files (DICT1.DAT, DICT2.DAT, and DICT3.DAT) should be deleted and BUILDDICT.COM should be re-executed.

Common problems that may develop when running EASIE.COM include: (a) no dictionary files in the current directory; (b) relation or template files not in the current directory; (c) relation or template already exists in the dictionary before EASIE.COM is run attempting to create that same relation or template; or (d) a template references a relation that has not been created.

```
$ @EASIE/OUTPUT=RUNEASIE.OUT
$ DIR
Directory DISK:[EASIE]

BUILD_EASIE.EXE;1   DATADB1.DAT;1       DATADB2.DAT;1       DATADB3.DAT;1
DICT1.DAT;1         DICT2.DAT;1         DICT3.DAT;1         EASIE.COM;1
EASIE.DAT;1         ITEMP.COM;1         ITEMP.FOR;1         ITEMP.REV;1
OTEMP.FOR;1         OTEMP.REV;1         RUNEASIE.OUT;1      SCHEMA.DAT;1
SORT.DAT;1

Total of 17 files.
```

A GETDATA routine is in ITEMP.FOR, and a PUTDATA routine is in OTEMP.FOR. The schema file is in SCHEMA.DAT. The directory listing shows the REVIEWER files and the three RIM database files

are present. The files EASIE.COM, RUNEASIE.OUT, and SCHEMA.DAT are often deleted at this point.

We now look at the REVIEWER loader ITEMP.COM. A familiarity with the REVIEWER commands (section 7) is needed to understand the contents of this file. The user must edit this file and change the zeroes to any value consistent with the variable's type. Variables of type CHARACTER must have values enclosed in double quotes. All lines in the file not beginning with M must be left in place. Any lines beginning with M can be deleted if desired. After the proper values have been placed in the file, this command procedure may be executed to load the database.

```
$ TYPE ITEMP.COM
$ DEFINE SYS$OUTPUT TRASH.DAT
$ REVIEW ITEMP
C REL_1
M XXX 0
M YYY(1,1,1) 0
M YYY(1,1,2) 0
M YYY(1,2,1) 0
M YYY(1,2,2) 0
M YYY(1,3,1) 0
M YYY(1,3,2) 0
M YYY(2,1,1) 0
M YYY(2,1,2) 0
M YYY(2,2,1) 0
M YYY(2,2,2) 0
M YYY(2,3,1) 0
M YYY(2,3,2) 0
C REL_2
M COUNT 0
E
$ DEASSIGN SYS$OUTPUT
$ DELETE TRASH.DAT;*
```

The SORT.DAT file is shown on the following page. This file may be created to help track down variables in the data base and can be very useful during program development and integration.

APPENDIX E

$ TYPE SORT.DAT

```
VARIABLE TYPE DIMENSION UNITS FORMAT DESCRIPTION
         RELATION_NAME - TEMPLATE_NAMES ...

COUNT    I 1                +                    I3      NUMBER OF PARTS
         REL_2      -    ITEMP
DESCR    C 80 1             +                    +       PROGRAM DESCRIPTION.
         REL_2      -    OTEMP ITABLE
VOL      R 1                M**3                 +       VOLUME OF OBJECT
         REL_2      -    OTEMP
XXX      R 1                FEET                 +       DISTANCE TO SATELLITE
         REL_1      -    ITEMP OTEMP
YYY      I 2,3,2            INCHES               I5      SIZE OF PARTS
         REL_1      -    ITEMP OTEMP
ZZZ      R 5,6              METERS               +       COORDINATES
         REL_1      -    OTEMP ITABLE

$
```

APPENDIX F
INSTRUCTIONS - EASIE CODING FORM

When one computer program provides data used by another program, it may be more effective to integrate these programs around a common information database and automate the data exchange rather than to manually perform the data exchange each time. These brief instructions explain which information about a computer program variable will be needed to integrate that program into an EASIE central database system using the utilties available for this purpose. Since the application programmer, who is the expert on the definition of variables needed in the database, may not perform the actual integration, he may instead provide the needed information on this form so that another program implementer can do the job for him.

In reference to the EASIE coding form, columns 2 through 6, 9, and 10 must be filled in by the application programmer who can define for the candidate program all input and output variables (names, descriptions, and units) that will be shared with the central database. Obviously, all input variables should be included, but not all variables typically written to a formatted output file will be deemed important for the database. The program expert can optionally fill in columns 1, 7, and 8 if such information is helpful. (See explanation for these columns below.)

Once all program inputs and any needed outputs are described on this coding form, the program implementer, who must be familiar with data already in the database, will fill in columns 11, 12, and 13. The implementer will determine whether the variables described already exist in the database and where (in which database relation) new variables should be placed. All of the information in columns 2 through 10 will then be used by the implementer for building relations, database I/O subroutines, etc.

A brief description of each column follows:

1. Column 1 (optional) provides a space for cross-referencing symbols used in program documentation with the program FORTRAN variables (column 2) used in the code. This is purely for the convenience of the application programmer and is intended to aid his bookkeeping.

2. Column 2 (required) is for listing all variables that are input to the program and all output variables selected for inclusion in the database. [Column 9 indicates how (I-input, O-output, and B-both) the variables are used.] Do not use local or subroutine variable names, but use only the variable name of the main program or the program module into which the data will be read/written from/to the database.

3. Column 3 (required) specifies whether the variable is real (R), integer (I), double precision (D), or character (CXXX). When it is type character, then the length of the character variable should be stated.

4. If the variable is subscripted, column 4 should declare the dimensions of the array (up to three-dimensional arrays are allowed).

5. Column 5 (required) is the description of the variable, up to 80 characters, but should be given concisely in the first 16 characters of the field which the REVIEWER automatically displays for Parameter type

only. If groups of input variables are somehow associated with various input options, then some indication of this grouping should also be included in the first 16 characters. Characters 17-80 will not be automatically displayed, but can be examined by the Expand line # (X menu selection) option provided by the REVIEWER.

6. Column 6 (required) gives the units of the variables which can be expressed in any understandable fashion up to 16 characters. The REVIEWER automatically displays 10 characters for Parameter Type only. However, a simple set of one- or two-character abbreviations will facilitate the possible later development of a units conversion utility. Examples of some abbreviations are:

FT	feet	N-M	newton meters
FT2	feet squared	NM	nautical miles
LB	pounds	LB/DY	pounds per day
DR	degrees Rankine		

7. Column 7 (optional) can be used if the program expert wishes to see a variable displayed in a particular way when using the REVIEWER.

8. Column 8 (optional) will allow the specification of a reference value for the variable. Such reference values could define a standard check case or typical run which would thus provide a coherent set of initial values for the database. Typically, at least the input values should be provided with reference values. If no values are given, data will be initialized in the database to zero or blank for character data.

9. Column 9 (required) indicates whether a variable is an input (I) to the program or an output (O) from the program, or it might be both (B).

10. Column 10 (if applicable) indicates if a variable is in a common block by giving the common block name.

    Columns 11, 12, and 13 are not filled in by the application programmer, but are used by the program implementer.

11. Column 11 is the attribute or parameter name. If the variable already exists in the database by another name, the name as it already appears in the database must be used and will be given in this column.

12. Column 12 contains the relation name. If the variable is new to the database, it may be placed in an existing relation or a new relation.

13. Column 13 contains the assignment of the relation type (A-ATTRIBUTE or P-PARAMETER).

    Note: Please fill in the program name and contact person's name and phone number in the spaces provided at the top of the form.

EASIE CODING FORM	VARIABLE DEFINITION AND USE	PAGE OF PAGES	DATE
DESIGN SYSTEM	APPLICATION PROGRAM	CONTACT PERSON PHONE	

1	2	3	4	5	6	7	8	9	10	11	12	13
REFERENCE DOCUMENT SYMBOL	VARIABLE NAME (8 CHAR.)	TYPE*	ARRAY DIMENSION (X,X,X)	VARIABLE DESCRIPTION (80 CHARACTERS)* (First 16 characters should be most descriptive)	UNITS* (16 CHAR.)	DISPLAY FORMAT (16 CHAR.)	DEFAULT VALUE	USE*	COMMON BLOCK NAME	ATTRIBUTE/ PARAMETER NAME	RELATION NAME (8 CHAR.)	TYPE*

NASA EASIE FORM 1
JULY 1987

* See Instructions for restrictions

SSD - SAB E - 1

APPENDIX G
VAX SYMBOL DEFINITIONS

The following VAX logical and symbol definitions are used throughout this document:

(1) TOAIDE - A logical defining the directory location of the EASIE software.

(2) RUNDICT - A symbol used to execute the SYSTEM LIBRARY PROCESSOR.

(3) RUNRIM - A symbol used to execute interactive RIM.

(4) LOADRIM - A logical defining the directory location of the RIM FORTRAN library.

(5) REVIEW - A symbol used to execute the REVIEWER.

These logicals and symbols are defined at the operating system level during installation of EASIE [5].

REFERENCES

1. Date, C. J.: The Systems Programming Series Volume I - AN INTRODUCTION TO DATABASE SYSTEMS, Third Edition, Addison-Wesley Publishing Company, February 1982.

2. Boeing Computer Services Company, BCS RIM - Relational Information Management System Version 6.0 User Guide, May 1983.

3. Randall, D. P.; Jones, K. H.; and Rowell, L. F.: The Environment For Application Software Integration and Execution (EASIE) Version 1.0. VOLUME IV - SYSTEM MAINTENANCE GUIDE. NASA TM-100576, April 1988.

4. Schwing, Dr. J. L.; Rowell, L. F.; and Criste, R. E.: The Environment For Application Software Integration and Execution (EASIE) Version 1.0. VOLUME III - PROGRAM EXECUTION GUIDE. NASA TM-100575, April 1988.

5. Rowell, L. F.; and Davis, J. S.: The Environment For Application Software Integration and Execution (EASIE) Version 1.0 VOLUME I - EXECUTIVE OVERVIEW. NASA TM-100573, May 1988.

6. Dube, R. P.; and Smith, M. R.: "Managing Geometric Information With A Database Management System", IEEE Computer Graphics and Applications, V. 3, No. 7, pp. 57-62, October 1983.

7. Jacky, J. P.; and KaPet, I. J.: "A General Purpose Data Entry Program", CACM, V. 26, No. 6, pp. 409-417, June 1983.

8. Structural Dynamics Research Corporation, I-DEAS™ USER'S GUIDE, Level 3, 5201.004, March 1986.

NASA

Report Documentation Page

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
NASA TM-100574		

4. Title and Subtitle

The Environment For Application Software Integration and Execution (EASIE) Version 1.0, Volume II - Program Integration Guide

5. Report Date

December 1988

6. Performing Organization Code

7. Author(s)

Kennie H. Jones, Donald P. Randall, Scott S. Stallcup, and Lawrence F. Rowell

8. Performing Organization Report No.

10. Work Unit No.

506-49-31-01

9. Performing Organization Name and Address

NASA Langley Research Center,
Hampton, VA 23665-5225

11. Contract or Grant No.

13. Type of Report and Period Covered

Technical Memorandum

12. Sponsoring Agency Name and Address

National Aeronautics and Space Administration
Washington, DC 20546-0001

14. Sponsoring Agency Code

15. Supplementary Notes

Kennie H. Jones, Donald P. Randall, and Scott S. Stallcup: Computer Sciences Corporation, Hampton, Virginia.
Lawrence F. Rowell: Langley Research Center, Hampton, Virginia.

16. Abstract

The Environment For Application Software Integration and Execution, EASIE, provides a methodology and a set of software utility programs to ease the task of coordinating engineering design and analysis codes. EASIE was designed to meet the needs of conceptual design engineers that face the task of integrating many stand-alone engineering analysis programs. Using EASIE, programs are integrated through a relational database management system.

Volume II, describes the use of a SYSTEM LIBRARY PROCESSOR to construct a DATA DICTIONARY describing all relations defined in the database, and a TEMPLATE LIBRARY. A TEMPLATE is a description of all subsets of relations (including conditional selection criteria and sorting specifications) to be accessed as input or output for a given application. Together, these form the SYSTEM LIBRARY which is used to automatically produce the database schema, FORTRAN subroutines to retrieve/store data from/to the database, and instructions to a generic REVIEWER program providing review/modification of data for a given template. Automation of these functions eliminates much of the tedious, error-prone work required by the conventional approach to database integration.

17. Key Words (Suggested by Author(s))		18. Distribution Statement
Program Interfacing	REVIEWER	Unclassified - Unlimited
Program Integration	FORMATTER	Subject category - 61
Database Management	EASIE	
Data Dictionary		

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of pages	22. Price
Unclassified	Unclassified	120	A06

NASA FORM 1626 OCT 86

```
ENTER DIRECTORY LOCATION FOR REVIEWER INPUT FILES
OR <CR> TO CREATE A LOCAL REVIEWER FILE
<CR>

ENTER TEMPLATE NAME
   OR <CR> TO CREATE REVIEWER FILES FOR ALL TEMPLATES
   OR "Q" TO QUIT
MAKGEOIN<CR>


SELECT OPTION:
BR  - TO BUILD RELATIONS
AR  - TO ADD TO AN EXISTING RELATION
BT  - TO BUILD A TEMPLATE
BF  - TO BUILD FORMATTER ROUTINES
BS  - TO BUILD A SCHEMA DUMP FOR DATA BASE
BRV - TO BUILD A REVIEWER INPUT FILE
BP  - TO BUILD PROGRAM DESCRIPTION
PT  - TO PRINT A TEMPLATE
LR  - TO LIST RELATIONS
PR  - TO PRINT A RELATION
X   - TO EXIT
BRV<CR>

ENTER DIRECTORY LOCATION FOR REVIEWER INPUT FILES
OR <CR> TO CREATE A LOCAL REVIEWER FILE
<CR>

ENTER TEMPLATE NAME
   OR <CR> TO CREATE REVIEWER FILES FOR ALL TEMPLATES
   OR "Q" TO QUIT
DRAWIN<CR>

ROW MODIFICATION FLAG FOR RELATION NODES   :
0 - FULL MODIFICATION
1 - MODIFY BUT CANNOT ADD ROWS
2 - CANNOT MODIFY ROWS
2<CR>

ROW MODIFICATION FLAG FOR RELATION FACES   :
0 - FULL MODIFICATION
1 - MODIFY BUT CANNOT ADD ROWS
2 - CANNOT MODIFY ROWS
2<CR>
```

```
SELECT OPTION:
BR  - TO BUILD RELATIONS
AR  - TO ADD TO AN EXISTING RELATION
BT  - TO BUILD A TEMPLATE
BF  - TO BUILD FORMATTER ROUTINES
BS  - TO BUILD A SCHEMA DUMP FOR DATA BASE
BRV - TO BUILD A REVIEWER INPUT FILE
BP  - TO BUILD PROGRAM DESCRIPTION
PT  - TO PRINT A TEMPLATE
LR  - TO LIST RELATIONS
PR  - TO PRINT A RELATION
X   - TO EXIT
X<CR>
```